

1994

Automating ban logic

Anish Mathuria
University of Wollongong

Follow this and additional works at: <https://ro.uow.edu.au/theses>

University of Wollongong

Copyright Warning

You may print or download ONE copy of this document for the purpose of your own research or study. The University does not authorise you to copy, communicate or otherwise make available electronically to any other person any copyright material contained on this site.

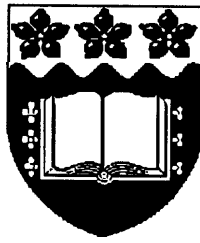
You are reminded of the following: This work is copyright. Apart from any use permitted under the Copyright Act 1968, no part of this work may be reproduced by any process, nor may any other exclusive right be exercised, without the permission of the author. Copyright owners are entitled to take legal action against persons who infringe their copyright. A reproduction of material that is protected by copyright may be a copyright infringement. A court may impose penalties and award damages in relation to offences and infringements relating to copyright material.

Higher penalties may apply, and higher damages may be awarded, for offences and infringements involving the conversion of material into digital or electronic form.

Unless otherwise indicated, the views expressed in this thesis are those of the author and do not necessarily represent the views of the University of Wollongong.

Recommended Citation

Mathuria, Anish, Automating ban logic, Master of Science (Hons.) thesis, Department of Computer Science, University of Wollongong, 1994. <https://ro.uow.edu.au/theses/2795>



AUTOMATING BAN LOGIC

A thesis submitted in partial fulfilment of the
requirements for the award of the degree

Master of Science (Honours)

from



UNIVERSITY OF WOLLONGONG

by

Anish Mathuria, B.E (Hons)

Department of Computer Science

1994

Acknowledgments

I am grateful to my supervisors Dr. Rei Safavi-Naini and Dr. Peter Nickolas for giving me an opportunity to undertake this work. Their constant support and guidance enabled timely completion of this project. I also benefited from discussions with the members of the Center for Computer Security Research in the Department of Computer Science. I am thankful to the administrative and support staff in the department and the Center for their help.

I would also like to express my gratitude to my parents for their understanding and continuous support during my studies.

Abstract

BAN Logic is a well-known formalism for the analysis of protocols used for authentication in distributed systems. This formalism deals with the evolution of beliefs of trustworthy principals executing an authentication protocol. The evolution of beliefs as a consequence of communication between the principals is modeled by the inference rules of the logic. The logic has been successful in formally discovering flaws and redundancies in a number of widely used protocols. A probabilistic extension of the logic allows calculation of a measure of trust in the goal of a protocol.

This thesis proposes a Prolog program to automate BAN logic analysis of authentication protocols. The program provides a protocol-independent inference engine, which takes as input the specification of a protocol, and generates all the logical statements describing the state of the principals executing the protocol. A proof explanation facility in the program provides a stepwise description of all minimal proofs of the logical statements attained during a protocol run. The program enables automatic identification of redundant assumptions and also facilitates probabilistic analysis of protocols. It has been used to perform machine-aided BAN logic analysis of several well-known protocols.

Publications arising out of this thesis

1. A. Mathuria, R. Safavi-Naini, and P. Nickolas, “Exploring Minimal BAN Logic Proofs of Authentication Protocols,” in *Proceedings of the Tenth International Conference on Information Security (IFIP SEC’94)*, May 1994.

Contents

1	Authentication Protocols	8
1.1	Introduction	8
1.2	The Needham-Schroeder Shared-Key Protocol	9
1.2.1	Messages	10
1.2.2	The Flaw	11
1.2.3	Fixing The Flaw	11
1.3	Formal Analysis of Protocols	13
2	BAN Logic	15
2.1	The Formalism	15
2.1.1	Syntax	15
2.1.2	Inference Rules	17
2.2	Protocol Analysis	20
2.2.1	Idealization	20
2.2.2	Assumptions and Goal	21
2.2.3	Annotation	22
2.3	Semantics	23
2.4	Proofs	26
2.5	Example Analysis	26
2.6	Protocol Verification	31
3	Machine-aided BAN Analysis	32
3.1	Introduction	32
3.2	The Protocol Analyzer	34
3.3	The Prolog Program	35

3.3.1	Structures	35
3.3.2	Predicates	36
3.4	The Needham-Schroeder Shared-Key Protocol	44
3.4.1	Program Representation	44
3.4.2	Open-ended Analysis	45
3.4.3	Redundant Assumptions	49
3.5	Remarks	50
4	Probabilistic BAN Analysis	51
4.1	Introduction	51
4.2	Probabilistic Logic	51
4.2.1	Probability of Derived Formula	52
4.2.2	Protocol Analysis	53
4.3	The Needham-Schroeder Shared-Key Protocol	56
4.3.1	Minimal Proofs	57
4.3.2	Lower Bounds	58
4.3.3	Interpreting the Bounds	60
5	Conclusions	62
5.1	Scope	62
A	References	64
B	Examples of Machine-aided BAN Analysis	67
B.1	The Otway-Rees Protocol	67
B.1.1	Program Representation	67
B.1.2	Proofs	68
B.2	The Needham-Schroeder Shared-Key Protocol	70
B.2.1	Program Representation	70
B.2.2	Proofs	70
B.3	The Kerberos Protocol	74
B.3.1	Program Representation	74
B.3.2	Proofs	75
B.4	The Yahalom Protocol	77

B.4.1	Program Representation	77
B.4.2	Proofs	78
B.5	The Needham-Schroeder Public-Key Protocol	81
B.5.1	Program Representation	81
B.5.2	Proofs	82
B.6	The CCITT.X509 Protocol	84
B.6.1	Program Representation	84
B.6.2	Proofs	84
C	BAN Logic Analyzer Program	87

List of Figures

1.1	The Needham-Schroeder Shared-Key Protocol.	10
3.1	Protocol Analysis.	34

Chapter 1

Authentication Protocols

This chapter gives a brief introduction to the identity authentication problem in distributed computer systems. The use of encryption for authentication is illustrated by means of the well-known Needham-Schroeder shared-key protocol [16]. Finally, the need for protocol verification is motivated by a discussion of the flaw in this protocol.

1.1 Introduction

A distributed computing system consists of various principals which communicate over a network. In such a system, the same communication lines are used by the principals to communicate with each other. This allows each principal to eavesdrop on the messages transmitted over the network. Each principal can also initiate communications and block, insert and alter messages which are sent over the network. Moreover, such mischief can be carried out by a legitimate user of the system itself. It is therefore obvious that the principals intending to communicate with each other in such a hostile environment would be suspicious about each other's identities.

Identity authentication refers to the process of mutual verification of identities by the principals wishing to engage in a secure communication. It involves checking of identities claimed by various principals and often precedes subsequent communications protected by shared/public keys. In a shared-key system, principals typically verify each other's identities by an authenticated

exchange of session keys. Such an exchange involves establishment of a new shared secret key or verifying a previously established shared key for further use. Verification of identities amongst principals in a public-key system typically involves acquiring each other's public keys.

An authentication protocol is an algorithm specifying a procedure for attaining authentication. It consists of a sequence of messages to be exchanged and the rules governing their exchange between the parties involved. The most common way of achieving an authenticated exchange of session/public-keys to attain authentication is by using a trusted principal known as the authentication server. Each principal of a shared-key system is assumed to share a secret key with the server. In a public-key system, it is assumed that each principal has registered his public key with the server whose public key is well-known. A typical authentication protocol makes use of nonces or timestamps to prevent replays of old messages from being accepted as recent. Nonces are fresh quantities in the sense that they have never been used for this purpose in the past.

A protocol can fail to achieve its goal due to a weakness in the message structure or the rules comprising the protocol rather than that of the underlying cryptosystem used. This is clearly demonstrated by the existence of flaws in a large number of published protocols [1]. The Needham-Schroeder shared-key protocol discussed in the next section is an instructive example of a flawed protocol.

1.2 The Needham-Schroeder Shared-Key Protocol

The aim of this protocol [16] is to securely distribute a session key K_{ab} to principals A and B . These principals are assumed to share secret keys K_{as} and K_{bs} with the server S respectively. The notation $A \rightarrow B : M$ denotes the communication of message M from A to B . A message M encrypted with the key K is denoted as $\{M\}_K$. The concatenation of messages M and M' is denoted by M, M' . The protocol is illustrated in Figure 1.1.

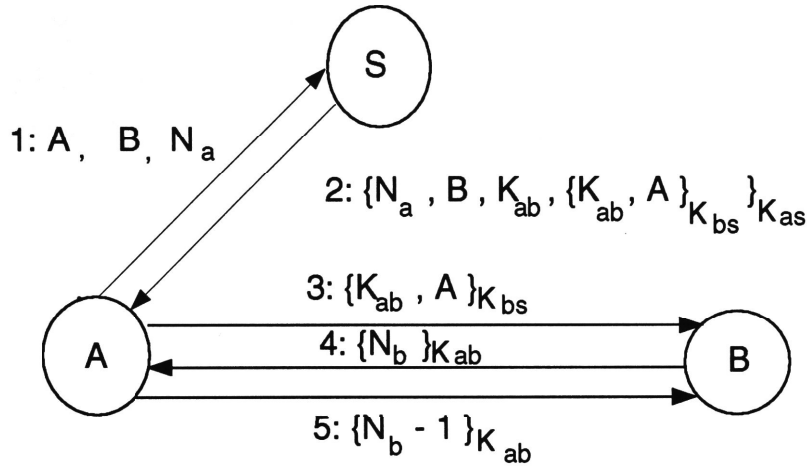


Figure 1.1: The Needham-Schroeder Shared-Key Protocol.

1.2.1 Messages

In the beginning of the protocol, the initiator, principal A , generates a nonce N_a and includes it in his request to the authentication server S for a secure channel to communicate with principal B by sending the following message:

$$A \rightarrow S : A, B, N_a \quad (1.1)$$

In response to (1.1), S sends to A a message conveying the session key K_{ab} and an authenticator for B :

$$S \rightarrow A : \{N_a, B, K_{ab}, \{K_{ab}, A\}_{K_{bs}}\}_{K_{as}} \quad (1.2)$$

Principal A decrypts (1.2) and checks for the presence of the nonce N_a and the identity of the intended recipient, principal B . The session key K_{ab} generated by S is such that it has never been used in the past. The freshness of the nonce N_a ensures that an intruder is not able to force reuse of a previously used session key by replaying a previously recorded message from S to A . Also, the identifier for B prevents an attacker X from impersonating B by altering B to X in (1.1). After verifying his nonce and the identity of the intended recipient, A forwards the authenticator $\{K_{ab}, A\}_{K_{bs}}$ to B :

$$A \rightarrow B : \{K_{ab}, A\}_{K_{bs}} \quad (1.3)$$

Principal B decrypts (1.3) to obtain the session key and the identity of the intending correspondent, principal A . He then generates a nonce N_b and requests a handshake with A to ensure that A is in possession of the session key K_{ab} :

$$B \rightarrow A : \{N_b\}_{K_{ab}} \quad (1.4)$$

Finally, A is expected to reply to B by returning $f(N_b) = N_b - 1$ encrypted with the session key:

$$A \rightarrow B : \{N_b - 1\}_{K_{ab}} \quad (1.5)$$

The last message (1.5) ends the protocol, and its execution is intended to leave principals A and B in possession of the session key K_{ab} .

1.2.2 The Flaw

This protocol has a serious flaw. The outline of the attack is as follows. An intruder X prevents message (1.3) from reaching B by blocking it and then emits an old authenticator containing a compromised session key CK :

$$X \rightarrow B : \{CK, A\}_{K_{bs}} \quad (1.6)$$

Message (1.6) is the old authenticator recorded by X during an earlier run of the protocol between A and B . The rest of the protocol proceeds as before with X masquerading as A . Thus B would unknowingly communicate with X instead of A .

1.2.3 Fixing The Flaw

Denning and Sacco [15] first pointed out the flaw in the Needham-Schroeder shared-key protocol by outlining the attack given above. They proposed a solution to repair the flaw in the original protocol by using timestamps. Subsequently, Needham and Schroeder [12] amended their protocol to eliminate the flaw without using timestamps. These two different solutions to fix the same flaw are described below.

Using Timestamps

In the Denning-Sacco protocol, the principals A and B do not generate nonces, unlike the original Needham-Schroeder protocol described in Section 1.2.1. Instead, the server S includes a timestamp T along with the session key K_{ab} in Message 2. The Denning-Sacco protocol [15] is:

Message 1 $A \rightarrow S : A, B$
Message 2 $S \rightarrow A : \{B, K_{ab}, T, \{A, K_{ab}, T\}_{K_{bs}}\}_{K_{as}}$
Message 3 $A \rightarrow B : \{A, K_{ab}, T\}_{K_{bs}}$

Principals A and B verify that the messages received by them are not replays by checking the timestamp T . This requires that the clocks of the communicating principals be synchronized.

Without Timestamps

The modified Needham-Schroeder protocol begins by the initiator, principal A , identifying himself to B :

$$A \rightarrow B : A \tag{1.7}$$

Upon receiving the identity of the intending correspondent in message (1.7), principal B forwards to him a message intended for S containing a nonce J :

$$B \rightarrow A : \{A, J\}_{K_{bs}} \tag{1.8}$$

Principal A includes message (1.8) in its request to S :

$$A \rightarrow S : A, B, N_a, \{A, J\}_{K_{bs}} \tag{1.9}$$

The server S decrypts the last component of message (1.9) and checks that the principal identified by B as the intending correspondent matches with the initiator, principal A . The server S then sends the following message to A :

$$S \rightarrow A : \{N_a, B, K_{ab}, \{K_{ab}, A, J\}_{K_{bs}}\}_{K_{as}} \quad (1.10)$$

Principal A decrypts message (1.10) and verifies its contents as in the original protocol and then forwards the part intended for B :

$$A \rightarrow B : \{K_{ab}, A, J\}_{K_{bs}} \quad (1.11)$$

Principal B is now ensured about the timeliness of the authenticator, unlike in the original protocol, due to the presence of the nonce J . The rest of the protocol proceeds as before.

1.3 Formal Analysis of Protocols

A large number of authentication protocols exist. Other examples of well-known protocols are the Needham-Schroeder public-key protocol [16], Otway-Rees protocol [11], Kerberos protocol [13], Andrew secure RPC handshake [14], Yahalom protocol [1], and the CCITT X.509 protocol [10]. The flawed Needham-Schroeder shared-key protocol and the revised protocols described above highlight the variety possible when designing authentication protocols. The wide variety of protocols existing in practice exemplify the need for a technique to determine whether a protocol functions correctly or not and to investigate the nature of guarantees offered by different protocols.

The main goal in formally analyzing a protocol is to ascertain the correctness of the protocol. Protocols are typically described by listing the contents of the protocol messages along with their informal interpretation. In such an informal interpretation, there is a danger of hidden assumptions being left out, which can have serious consequences for protocol security. One of the goals of formal analysis is to explicate the assumptions required for a given protocol to achieve its intended goal. In addition, many protocols contain redundancies, typically in the form of unnecessary encryption. Formal analysis can also help identify redundancies, which can be eliminated, thereby enhancing protocol efficiency.

The logic of authentication due to Burrow, Abadi and Needham (henceforth referred to as the BAN logic) [1] is a formal calculus aimed at modelling the central concepts involved in authentication. It is the most well-known and widely used formalism for analyzing authentication protocols [2]. The logic has been successful in formally discovering flaws and redundancies in several protocols [1]. The next chapter describes the logic and illustrates its use by an example analysis.

Chapter 2

BAN Logic

The BAN logic [1] formalizes the notions of trust and timeliness used in authentication protocols. This chapter describes the syntax and semantics of BAN logic [1]. The application of this logic is demonstrated by an analysis of the Needham-Schroeder shared-key protocol.

2.1 The Formalism

BAN is a modal logic of belief. The logic has no explicit notion of time. Two epochs are considered to distinguish the time associated with messages. The present epoch begins with the opening message and ends with the last message of the protocol run under consideration. All messages sent before the present epoch are considered to be in the past. Negation is not provided in the logic. An inference made during a particular run holds for the entirety of that run. The syntax and inference rules of the BAN logic are given below.

2.1.1 Syntax

Protocol messages are represented by *formulae* which are also referred to as *statements*. Typically, the symbols A, B , and S denote principals, K_{ab}, K_{as} , and K_{bs} denote shared keys, K_a, K_b , and K_s denote public keys, K_a^{-1}, K_b^{-1} , and K_s^{-1} denote the corresponding private keys, and N_a, N_b , and N_c denote nonces. The symbols P, Q , etc. range over principals, K ranges over encryp-

tion keys, and X, Y , etc. range over formulas.

Formulae

The formulae of the logic are listed below together with their operational meanings.

- $P \models X$: P believes X . This is the principal construct of the logic.
- $P \triangleleft X$: P sees X . P has received a message X and is able to repeat it in other messages.
- $P \vdash X$: P once said X . P uttered X either in the past or present.
- $P \models\Rightarrow X$: P has jurisdiction over X . This construct is used to represent the notions of trust and delegation.
- $\sharp(X)$: X is fresh; i.e., X has never appeared in any message sent in the past.
- $P \stackrel{K}{\leftrightarrow} Q$: P and Q may use the good shared key K to communicate. The key K will never be discovered by anyone except P or Q , or a principal trusted by either of them.
- $\stackrel{K}{\mapsto} P$: K is the public key of P . The conjugate private key is denoted by K^{-1} .
- $P \stackrel{X}{\rightleftharpoons} Q$: The shared secret X is known only to P and Q , and to principals trusted by them. It may be used by P and Q to prove their identities to one another.
- $\{X\}_K$: X encrypted with key K . It is an abbreviation for the longer expression $\{X\}_K \text{ from } P$, where P denotes the originator of $\{X\}_K$.
- $\langle X \rangle_Y$: X combined with Y . The formula Y is a secret and serves as a proof of origin for X .

In addition, the conjunction of X and Y , written as (X, Y) , is also a formula. The properties of associativity and commutativity hold for conjunctions, which are treated as sets.

2.1.2 Inference Rules

The inference rules of the logic formalize reasoning about authentication protocols. They specify relationships holding between formulae of the forms $P \models X$ and $P \triangleleft X$.

An inference rule is written in the form:

$$\frac{X_1, \dots, X_n}{Y}$$

Such a rule states that if X_1 and ... and X_n hold, then Y holds. In the inference rules given below, some rules are written adjacent to each other for the sake of convenience.

- **Message-meaning rules**

The message-meaning rules deal with the interpretation of messages. There are three message-meaning rules, as follows.

For shared keys:

$$\frac{P \models Q \stackrel{K}{\leftrightarrow} P, P \triangleleft \{X\}_K}{P \models Q \sim X}$$

i.e., if P believes that he shares the key K with Q and sees a message X encrypted with K , then P believes that Q sent X .

For public keys:

$$\frac{P \models \stackrel{K}{\mapsto} Q, P \triangleleft \{X\}_{K^{-1}}}{P \models Q \sim X}$$

i.e., if P believes that K is the public key of Q and sees a message X encrypted with the private key K^{-1} of Q , then P believes that Q sent X .

For shared secrets:

$$\frac{P \models Q \stackrel{Y}{=} P, P \triangleleft \langle X \rangle_Y}{P \models Q \sim X}$$

i.e., if P believes that he shares the secret Y with Q and sees $\langle X \rangle_Y$, then P believes that Q sent X .

- **Nonce-verification rule**

The sender of a recent message is presumed to believe in it:

$$\frac{P \models \sharp(X), P \models Q \vdash X}{P \models Q \models X}$$

i.e., if P believes that Q sent X in the current run, then P believes that Q believes X .

- **Jurisdiction rule**

A principal trusts an authority on matters delegated to the authority:

$$\frac{P \models Q \models X, P \models Q \models X}{P \models X}$$

i.e., if P believes that Q has jurisdiction over X and believes that Q believes X , then P believes X .

- **Belief rules**

These rules characterize the property of the belief operator:

$$\frac{P \models X, P \models Y}{P \models (X, Y)} \quad \frac{P \models (X, Y)}{P \models X} \quad \frac{P \models Q \models (X, Y)}{P \models Q \models X}$$

- **Utterance rule**

This rule characterizes the property of the “once said” operator:

$$\frac{P \models Q \vdash (X, Y)}{P \models Q \vdash X}$$

- **Sight rules**

This rule characterizes the property of the “sees” operator:

$$\frac{P \triangleleft (X, Y)}{P \triangleleft X} \quad \frac{P \triangleleft \langle X \rangle_Y}{P \triangleleft X}$$

- **Message decryption rules**

If a principal receives a message encrypted with a key he believes to be good for communicating with someone, then he sees the decrypted message:

$$\frac{P \models Q \xrightarrow{K} P, P \triangleleft \{X\}_K}{P \triangleleft X}$$

If a principal receives a message encrypted with his public key, then he sees the decrypted message:

$$\frac{P \models \xrightarrow{K} P, P \triangleleft \{X\}_K}{P \triangleleft X}$$

If a principal believes that K is the public key of someone and sees a message encrypted with the corresponding private key K^{-1} , then he sees the decrypted message:

$$\frac{P \models \xrightarrow{K} Q, P \triangleleft \{X\}_{K^{-1}}}{P \triangleleft X}$$

- **Freshness rule**

Freshness of at least one component of a formula implies that the whole formula is fresh:

$$\frac{P \models \#(X)}{P \models \#(X, Y)}$$

- **Bidirectionality rules**

A shared key or secret can be used between a pair of principals in either direction. The following rules reflect this symmetry:

$$\frac{P \models R \xrightarrow{K} R'}{P \models R' \xrightarrow{K} R} \quad \frac{P \models Q \models R \xrightarrow{K} R'}{P \models Q \models R' \xrightarrow{K} R}$$

$$\frac{P \models R \xrightarrow{K} R'}{P \models R' \xrightarrow{K} R} \quad \frac{P \models Q \models R \xrightarrow{K} R'}{P \models Q \models R' \xrightarrow{K} R}$$

In the message decryption rule for shared keys, P can only use shared keys believed to be good for communicating with some specified principal. This inference rule can be made more liberal by allowing for decryption with

shared keys that are not known to be good. An additional inference rule of the logic provides such a mechanism:

$$\frac{P \models R \sim Q \stackrel{K}{\leftrightarrow} P, P \triangleleft \{X\}_K}{P \triangleleft X}$$

i.e., P may use any key mentioned to him to decrypt the contents of an encrypted message.

2.2 Protocol Analysis

The main purpose in analyzing a protocol is to determine whether it achieves its intended goal. This requires the protocol itself and the initial assumptions and goal to be made explicit in the logic. The inference rules of the logic may then be applied to the assumptions and messages to determine the position attained by the principals. The three phases of protocol analysis in BAN logic are discussed below.

2.2.1 Idealization

Idealization is the process of transforming the informal description of a protocol into a form suitable for direct manipulation in the logic. In the idealized protocol, messages of the protocol are represented by corresponding logical formulae. An idealized protocol is a finite sequence of n “send” statements, S_1, \dots, S_n , each of the form $P \rightarrow Q : X$, where the formula X represents the message sent by P to Q .

The idealized protocol can be viewed as a clearer description of the protocol. It is obtained from the informal description of the protocol by using the following guidelines.

All messages sent in clear are omitted. The interpretation of the contents of the remaining messages is made explicit in the logic. In particular, the conditions required to hold for the messages to be sent are included in the idealized version. This requires understanding of the entire protocol. Protocol idealization cannot therefore be done by merely looking at each individual message in itself.

2.2.2 Assumptions and Goal

The assumptions describe the initial state of the principals and the goal specifies the intended outcome. The formulation of the assumptions and the goals of protocols in BAN logic is discussed below.

Assumptions

Principals in a protocol start with a set of initial beliefs. These initial beliefs form the assumptions of the protocol. Typically, the assumptions state the following:

- The keys initially shared between the principals
- The freshness of the nonces generated by various principals
- The trust the clients have in the server to generate and distribute keys

Goal

The goal of a protocol describes the requirement for mutual authentication between the principals. For shared-key protocols, the requirement is that the principals be in the possession of the session key at the end of the protocol. The secure establishment of the shared key K between principals A and B is formalized as:

$$A \models A \stackrel{K}{\leftrightarrow} B$$

$$B \models A \stackrel{K}{\leftrightarrow} B$$

These are referred to as first-order goals of A and B respectively. Some protocols are intended to further guarantee each principal that the other principal is in possession of the shared key. This requirement translates into beliefs about others' beliefs, called second-order goals:

$$A \models B \models A \stackrel{K}{\leftrightarrow} B$$

$$B \models A \models A \stackrel{K}{\leftrightarrow} B$$

In public-key protocols intended for the secure transfer of public keys, the requirement for mutual authentication is that the principals be in possession of the public key of the other principal at the end of the protocol run. The secure transfer of public keys between the principals A and B having public keys K_a and K_b respectively is formalized as:

$$A \models \overset{K_b}{\mapsto} B$$

$$B \models \overset{K_a}{\mapsto} A$$

The goals described above are the most commonly intended in protocols. Any other goals are generally obvious from the context.

2.2.3 Annotation

Once the protocol is idealized and the initial assumptions and goal of the protocol are made explicit, the analysis is carried out by annotating the idealized protocol with formulas and manipulating them with the inference rules. In the annotated protocol, assertions are inserted after each idealized message. An assertion is a set of formulas of the forms $P \models X$ and $P \triangleleft X$, that hold after the execution of the step they follow. The first assertion contains the initial assumptions and the last contains the conclusions:

$$\{assumptions\}S_1\{assertion\ 1\}\dots\{assertion\ n-1\}S_n\{conclusions\}$$

Annotation rules

The rules to derive legal annotations are as follows.

- The effect of executing a protocol step is that the sent message is seen by the recipient. All formulas that hold before the execution of this step hold after its execution:

$$\vdash \{Y\}(P \rightarrow Q : X)\{Y, Q \triangleleft X\}$$

- Annotations can be concatenated:

$$\frac{\vdash \{X\}S_1 \dots \{Y\}, \vdash \{Y\}S'_1 \dots \{Z\}}{\vdash \{X\}S_1 \dots \{Y\}S'_1 \dots \{Z\}}$$

- Intermediate assertions and conclusions can be weakened by applying the inference rules to derive new assertions from existing ones:

$$\frac{\vdash \{Y\}S_1 \dots \{X_1\} \dots \{X_i\} \dots \{X_n\}, X_i \vdash X'_i}{\vdash \{Y\}S_1 \dots \{X_1\} \dots \{X'_i\} \dots \{X_n\}}$$

- The consequences of the assumptions can always be made explicit:

$$\frac{\vdash \{X\}S \dots \{Y\}, X \vdash X'}{\vdash \{X, X'\}S \dots \{Y\}}$$

Let S denote the idealized protocol with the assumptions X and goal Y . The protocol is said to *achieve its goal* if the following theorem can be proved by using the annotation rules described above:

$$\vdash \{X\}S\{Y\}$$

The steps leading to the goal of mutual authentication typically involve application of the message-meaning, nonce-verification and jurisdiction inference rules. In practice, annotated protocols are not used explicitly during the analysis since only the assumptions and conclusions are useful for the purpose of protocol verification.

2.3 Semantics

The semantics of the logic [1] is operational, in that it specifies how principals attain beliefs by computation. Principals obtain new beliefs by applying the inference rules of the logic to their current beliefs.

The *local state* s_P of a principal P is characterized by two sets of formulae, \mathcal{M}_P and \mathcal{B}_P . The set \mathcal{M}_P contains as its elements the messages seen by P , and \mathcal{B}_P is the set of beliefs held by P . The closure properties of the sets

\mathcal{M}_P and \mathcal{B}_P correspond to the inference rules of the logic. For example, if $(P \xrightarrow{K} Q) \in \mathcal{B}_P$ and $\{X\}_K \in \mathcal{M}_P$ then $X \in \mathcal{M}_P$.

A *global state* is a collection of the local states of principals executing the protocol. Typically, it is a triple containing the local states of A , B , and S . The local state of P in a global state s is denoted as s_P , and the corresponding message and beliefs sets are denoted as $\mathcal{M}_P(s)$ and $\mathcal{B}_P(s)$ respectively. The *satisfaction* relation between s and the formulas $P \models X$, and $P \triangleleft X$ is defined as follows. The formula $P \models X$ is satisfied in s , or holds in s , if $X \in \mathcal{B}_P(s)$, and $P \triangleleft X$ holds in s if $X \in \mathcal{M}_P(s)$. A set of formulas holds in a given state if each of its members holds in that state.

A *run* is a finite sequence of global states s_0, \dots, s_n in which the message and belief sets increase monotonically. In other words, for each principal P , $\mathcal{B}_P(s_i) \subseteq \mathcal{B}_P(s_{i+1})$ and $\mathcal{M}_P(s_i) \subseteq \mathcal{M}_P(s_{i+1})$ for $0 \leq i \leq n-1$. A *run of a protocol*, $(P_1 \rightarrow Q_1 : X_1), \dots, (P_n \rightarrow Q_n : X_n)$, is a run of length $n+1$ in which all prescribed messages are communicated, i.e., $X_i \in \mathcal{M}_{Q_i}(s_i)$ for all $1 \leq i \leq n$. The initial state s_0 of the run corresponds to the beliefs and messages which may be present before execution of the first send statement.

An *annotation* for the protocol *holds in a protocol run* if all formulae in the annotation hold in the corresponding global states. It is *valid* if it holds in all runs of the protocol. The annotations rules given in Section 2.2.3 are sound and complete. This follows trivially by considering a protocol run in which only the prescribed messages are communicated. In such a run, all valid annotations must hold and conversely any annotation that holds in this run is derivable.

The notion of a state introduced above gives a meaning to the \models and \triangleleft operators. To give a meaning to the remaining operators, the notion of a state is extended as described below.

- In addition to \mathcal{B}_P and \mathcal{M}_P , the set \mathcal{O}_P of formulas once said by P is included in the local state of each principal P . The closure properties of \mathcal{O}_P are:

- if $(\{X\}_K \text{ from } P) \in \mathcal{O}_P$, then $X \in \mathcal{O}_P$
- if $\langle X \rangle_Y \in \mathcal{O}_P$, then $X \in \mathcal{O}_P$

– if $(X, Y) \in \mathcal{O}_P$, then $X \in \mathcal{O}_P$

In addition, if the i th action of a protocol is $P_i \rightarrow Q_i : X_i$, then the set of formulas once said increases only for P_i ; i.e., if $R \neq P_i$, then $\mathcal{O}_R(s_i) = \mathcal{O}_R(s_{i-1})$ and $\mathcal{O}_{P_i} = \mathcal{O}_{P_i} \cup \{X_i\}$. Also, each principal believes formulas he said recently; i.e., if $X \in \mathcal{O}_P(s)$, then $X \in \mathcal{B}_P$.

Finally, $P \sim X$ holds in state s if $X \in \mathcal{O}_P$.

- Each principal P has jurisdiction over a set of formulas \mathcal{J}_P . If $X \in \mathcal{J}_P$ and the belief $P \models X$ holds, then X also holds.

The states in the run satisfy $P \models X$ if $X \in \mathcal{J}_P$.

- Each run assigns a set of good shared keys $\mathcal{K}_{\{P,Q\}}$ to each pair of principals (P, Q) . Only appropriate principals use these keys; i.e., if $R \triangleleft (\{X\}_K \text{ from } R')$ holds and $K \in \mathcal{K}_{\{P,Q\}}$, then either $R' = P$ and $(\{X\}_K \text{ from } P) \in \mathcal{O}_P$ or $R' = Q$ and $(\{X\}_K \text{ from } Q) \in \mathcal{O}_Q$.

The states in the run satisfy $P \stackrel{K}{\leftrightarrow} Q$ if $K \in \mathcal{K}_{\{P,Q\}}$.

- Each run assigns a set of good public keys \mathcal{K}_P to each principal P . Only the appropriate principals can use the matching secret keys; i.e., if $R \triangleleft (\{X\}_{K^{-1}} \text{ from } R')$ holds and $K \in \mathcal{K}_P$, then $R' = P$ and $(\{X\}_{K^{-1}} \text{ from } P) \in \mathcal{O}_P$.

The states in the run satisfy $\stackrel{K}{\rightarrow} P$ if $K \in \mathcal{K}_P$.

- Each run assigns a set of shared secrets $\mathcal{S}_{\{P,Q\}}$ to each pair of principals (P, Q) . These shared secrets are used only by appropriate principals; i.e., if $R \triangleleft \langle X \rangle_Y$ holds and $Y \in \mathcal{S}_{\{P,Q\}}$, then either $\langle X \rangle_Y \in \mathcal{O}_P$ or $\langle X \rangle_Y \in \mathcal{O}_Q$.

The states in the run satisfy $P \stackrel{X}{\rightleftharpoons} Q$ if $X \in \mathcal{S}_{\{P,Q\}}$.

- Each run determines a set of fresh formulas \mathcal{F} . The closure property of this set corresponds to the freshness rule; i.e., if $X \in \mathcal{F}$ and X is a subformula of Y , then $Y \in \mathcal{F}$. Also, if $X \in \mathcal{F}$ and $X \in \mathcal{O}_P(s_i)$, then $X \ni \mathcal{O}_P(s_0)$.

The states in the run satisfy $\sharp(X)$ if $X \in \mathcal{F}$.

2.4 Proofs

It is useful to formalize the notions of a proof in the logic, following [4].

Let A be a finite set of assumptions. Define the *closure* of A , denoted as A^+ to be the set of all formulae derivable from A by applying the inference rules of the logic. Let R denote the set of rule instances used in deriving the formulae in A^+ and let $B = A \cup R$.

A *proof of a formula* $c \in A^+$ *from* B is a sequence of formulae $S = (s_1, \dots, s_n)$ such that s_n is c , and each s_j for $j < n$ is in B and is either an assumption in A or is inferred from previous formulae in the sequence by a rule instance in R .

The notion of equivalent proofs can be defined as follows. Let $S' = (s'_1, \dots, s'_{n'})$ be a proof of c from B . Then, S' is *equivalent* to S if $n = n'$ and s_1, \dots, s_{n-1} are a permutation of s'_1, \dots, s'_{n-1} .

Let $B = \{b_1, \dots, b_m\}$. The *index set* of a proof of c from B is a set $I \subseteq \{1, \dots, m\}$ such that $\{b_i : i \in I\}$ form the proof. Let I denote the index set of a proof S of c from B . The proof S is *minimal* if there is no proof S' of c from B whose index set I' is such that $I' \subset I$.

During protocol analysis, the goal of a principal is in general expressed as one or more logical formulae. Informally, a redundant assumption is a formula whose exclusion does not prevent the goal from being attained. This notion can be formalized in terms of the minimal proofs of the desired goal, as follows.

Let $G = \{s_1, \dots, s_n\}$ denote the desired goal of a principal. A *redundant assumption* of the principal is a formula s such that for every s_i , there exists at least one minimal proof of s_i , which does not contain s .

2.5 Example Analysis

This section presents the BAN logic analysis of the Needham-Schroeder shared-key protocol. This protocol has a serious flaw, as pointed out in the last chapter. The analysis given below demonstrates how BAN logic analysis is successful in discovering this flaw. The Needham-Schroeder shared-key

protocol is:

- Message 1 $A \rightarrow S : A, B, N_a$
- Message 2 $S \rightarrow A : \{N_a, B, K_{ab}, \{K_{ab}, A\}_{K_{bs}}\}_{K_{as}}$
- Message 3 $A \rightarrow B : \{K_{ab}, A\}_{K_{bs}}$
- Message 4 $B \rightarrow A : \{N_b\}_{K_{ab}}$
- Message 5 $A \rightarrow B : \{N_b - 1\}_{K_{ab}}$

Idealized Protocol

Following the guidelines given in Section 2.2.1, the idealized protocol obtained is:

- Message 2 $S \rightarrow A : \{N_a, A \xleftrightarrow{K_{ab}} B, \{A \xleftrightarrow{K_{ab}} B\}_{K_{bs}}\}_{K_{as}}$
- Message 3 $A \rightarrow B : \{A \xleftrightarrow{K_{ab}} B\}_{K_{bs}}$
- Message 4 $B \rightarrow A : \{N_b, A \xleftrightarrow{K_{ab}} B\}_{K_{ab}}$
- Message 5 $A \rightarrow B : \{N_b, A \xleftrightarrow{K_{ab}} B\}_{K_{ab}}$

The idealized protocol consists of only four messages instead of five in the original protocol. This is because the first message, which is in cleartext, is omitted. The idealization of message 2 is done as follows. The sequence B, K_{ab} in the original message is replaced by the formula $A \xleftrightarrow{K_{ab}} B$. This is because upon decrypting this message A finds the nonce N_a and the identifier of the intended recipient, principal B , both sent by A in his request to S and is thus assured about K_{ab} . Similarly, in the authenticator for B , the sequence K_{ab}, A is replaced by the formula $A \xleftrightarrow{K_{ab}} B$ since S certifies K_{ab} for B as well. The formula $A \xleftrightarrow{K_{ab}} B$ is included in the last two messages of the idealized protocol since they are meant to assure each principal that the other believes that K_{ab} is good.

Assumptions and Goal

The following assumptions describe the initial beliefs of the principals:

$$\begin{array}{ll}
A \models A \stackrel{K_{as}}{\leftrightarrow} S & B \models B \stackrel{K_{bs}}{\leftrightarrow} S \\
S \models A \stackrel{K_{as}}{\leftrightarrow} S & S \models B \stackrel{K_{bs}}{\leftrightarrow} S \\
S \models A \stackrel{K_{ab}}{\leftrightarrow} B &
\end{array}$$

$$A \models S \Rightarrow A \stackrel{K}{\leftrightarrow} B \quad B \models S \Rightarrow A \stackrel{K}{\leftrightarrow} B$$

$$A \models \#(N_a) \quad B \models \#(N_b)$$

The first four assumptions regarding the initial keys shared between the clients and the server are obvious. The assumption $S \models A \stackrel{K_{ab}}{\leftrightarrow} B$ states that S knows the session key. The next two assumptions indicate the trust the clients have in S to generate session keys. The last two assumptions state that the nonces are generated by principals who consider them to be fresh.

The aim of the protocol is to leave both clients in the possession of the shared session key and to assure them that the other is in possession of this key. The goal is thus formalized as:

$$\begin{array}{ll}
A \models A \stackrel{K_{ab}}{\leftrightarrow} B & B \models A \stackrel{K_{ab}}{\leftrightarrow} B \\
A \models B \models A \stackrel{K_{ab}}{\leftrightarrow} B & B \models A \models A \stackrel{K_{ab}}{\leftrightarrow} B
\end{array}$$

Analysis

Applying the message-meaning rule to Message 2 we have:

$$\frac{A \models A \stackrel{K_{as}}{\leftrightarrow} S, A \triangleleft \{N_a, A \stackrel{K_{ab}}{\leftrightarrow} B, \{A \stackrel{K_{ab}}{\leftrightarrow} B\}_{K_{bs}}\}_{K_{as}}}{A \models S \sim (N_a, A \stackrel{K_{ab}}{\leftrightarrow} B, \{A \stackrel{K_{ab}}{\leftrightarrow} B\}_{K_{bs}})}$$

The freshness of the message obtained above can be deduced by applying the freshness rule:

$$\frac{A \models \#(N_a)}{A \models \#(N_a, A \stackrel{K_{ab}}{\leftrightarrow} B, \{A \stackrel{K_{ab}}{\leftrightarrow} B\}_{K_{bs}})}$$

By applying the nonce-verification rule we have:

$$\frac{A \models \#(N_a, A \stackrel{K_{ab}}{\leftrightarrow} B, \{A \stackrel{K_{ab}}{\leftrightarrow} B\}_{K_{bs}}), A \models S \sim (N_a, A \stackrel{K_{ab}}{\leftrightarrow} B, \{A \stackrel{K_{ab}}{\leftrightarrow} B\}_{K_{bs}})}{A \models S \models (N_a, A \stackrel{K_{ab}}{\leftrightarrow} B, \{A \stackrel{K_{ab}}{\leftrightarrow} B\}_{K_{bs}})}$$

By applying the belief rule we have:

$$\frac{A \models S \models (N_a, A \overset{K_{ab}}{\leftrightarrow} B, \{A \overset{K_{ab}}{\leftrightarrow} B\}_{K_{bs}})}{A \models S \models A \overset{K_{ab}}{\leftrightarrow} B}$$

Finally, by applying the jurisdiction rule we obtain the first-order goal of A :

$$\frac{A \models S \models A \overset{K_{ab}}{\leftrightarrow} B, A \models S \models A \overset{K_{ab}}{\leftrightarrow} B}{A \models A \overset{K_{ab}}{\leftrightarrow} B}$$

Principal A then forwards the authenticator for B as Message 3. By applying the message-meaning rule we have:

$$\frac{B \models B \overset{K_{bs}}{\leftrightarrow} S, B \triangleleft \{A \overset{K_{ab}}{\leftrightarrow} B\}_{K_{bs}}}{B \models S \sim A \overset{K_{ab}}{\leftrightarrow} B}$$

Principal B cannot make further deductions to attain his first-order goal since he knows of nothing in the message sent by S to be fresh. The only way B can proceed further is to assume that the message sent by S is not a replay. This extra assumption can be stated as $B \models \#(A \overset{K_{ab}}{\leftrightarrow} B)$. Once this dubious assumption is made B can derive his first-order goal by applying the nonce-verification and jurisdiction rules as follows:

$$\frac{B \models \#(A \overset{K_{ab}}{\leftrightarrow} B), B \models S \sim A \overset{K_{ab}}{\leftrightarrow} B}{B \models S \models A \overset{K_{ab}}{\leftrightarrow} B}$$

$$\frac{B \models S \models A \overset{K_{ab}}{\leftrightarrow} B, B \models S \models A \overset{K_{ab}}{\leftrightarrow} B}{B \models A \overset{K_{ab}}{\leftrightarrow} B}$$

The nonce handshake at the end of the protocol is intended to assure each client that the other is in possession of the shared key. By applying the message-meaning rule to Message 4 we have:

$$\frac{A \models A \overset{K_{ab}}{\leftrightarrow} B, A \triangleleft \{N_b, A \overset{K_{ab}}{\leftrightarrow} B\}_{K_{ab}}}{A \models B \sim (N_b, A \overset{K_{ab}}{\leftrightarrow} B)}$$

Now, A is unable to proceed further to derive his second-order goal because he knows of nothing in the message sent by B to be fresh. This highlights the need for the server to always make sure that the key he generates

is fresh, and to tell A so. The first requirement is expressed as the additional assumptions $S \models \sharp(A \stackrel{K_{ab}}{\leftrightarrow} B)$ and $A \models S \vdash \sharp(A \stackrel{K_{ab}}{\leftrightarrow} B)$. The second requirement is incorporated by including the formula $\sharp(A \stackrel{K_{ab}}{\leftrightarrow} B)$ in the idealized Message 2. The modified idealized Message 2 is:

$$\text{Message 2 } S \rightarrow A : \{N_a, A \stackrel{K_{ab}}{\leftrightarrow} B, \sharp(A \stackrel{K_{ab}}{\leftrightarrow} B), \{A \stackrel{K_{ab}}{\leftrightarrow} B\}_{K_{bs}}\}_{K_{as}}$$

Once this is done, the previous deductions made by A by applying the message-meaning rule and the nonce-verification rule are modified to include the formula $\sharp(A \stackrel{K_{ab}}{\leftrightarrow} B)$ in the message sent by S . One more application of the belief rule leads to:

$$\frac{A \models S \models (N_a, A \stackrel{K_{ab}}{\leftrightarrow} B, \sharp(A \stackrel{K_{ab}}{\leftrightarrow} B), \{A \stackrel{K_{ab}}{\leftrightarrow} B\}_{K_{bs}})}{A \models S \models \sharp(A \stackrel{K_{ab}}{\leftrightarrow} B)}$$

The jurisdiction rule then enables A to deduce the freshness of K_{ab} :

$$\frac{A \models S \vdash \sharp(A \stackrel{K_{ab}}{\leftrightarrow} B), A \models S \models \sharp(A \stackrel{K_{ab}}{\leftrightarrow} B)}{A \models \sharp(A \stackrel{K_{ab}}{\leftrightarrow} B)}$$

Hence, A is now able to apply the nonce-verification rule to the message sent by B and obtain his second-order goal:

$$\frac{A \models \sharp(A \stackrel{K_{ab}}{\leftrightarrow} B), A \models B \vdash A \stackrel{K_{ab}}{\leftrightarrow} B}{A \models B \models A \stackrel{K_{ab}}{\leftrightarrow} B}$$

The derivation of the second-order goal of B proceeds as follows. Principal B applies the message-meaning rule to Message 5:

$$\frac{B \models A \stackrel{K_{ab}}{\leftrightarrow} B, B \triangleleft \{N_b, A \stackrel{K_{ab}}{\leftrightarrow} B\}_{K_{ab}}}{B \models A \vdash (N_b, A \stackrel{K_{ab}}{\leftrightarrow} B)}$$

By applying the freshness rule:

$$\frac{B \models \sharp(N_b)}{B \models \sharp(N_b, A \stackrel{K_{ab}}{\leftrightarrow} B)}$$

By applying the nonce-verification rule:

$$\frac{B \models \sharp(N_b, A \stackrel{K_{ab}}{\leftrightarrow} B), B \models A \vdash (N_b, A \stackrel{K_{ab}}{\leftrightarrow} B)}{B \models A \models (N_b, A \stackrel{K_{ab}}{\leftrightarrow} B)}$$

Finally, by applying the belief rule we obtain the second-order goal of B :

$$\frac{B \models A \models (N_b, A \stackrel{K_{ab}}{\leftrightarrow} B)}{B \models A \models A \stackrel{K_{ab}}{\leftrightarrow} B}$$

It is clear that once the dubious assumption is made, B can send anything, not necessarily a nonce, encrypted with K_{ab} as Message 4 and will still be able to attain his second-order goal. Hence the assumption $B \models \sharp(N_b)$ is redundant. Also, it can be verified that the protocol achieves its goal even if S does not encrypt the authenticator for B with K_{as} .

This completes the analysis of the protocol. The analysis shows how the flaw is manifested by the need to make a dubious assumption. It also brings out the requirement for the additional assumptions regarding the freshness of the session key in order for A to be able to achieve his second-order goal. It further points out the redundancy in the assumption set and shows that eliminating the double encryption in Message 2 does not weaken the protocol.

2.6 Protocol Verification

As seen by the example analysis presented above, protocol verification in BAN logic amounts to proving the desired goal by applying the inference rules to the assumptions and the idealized protocol. To start analyzing a protocol, the usual assumptions are made in the beginning and deductions possible using the inference rules are performed. Finally, the assumptions are refined until the goal is achieved. The protocol is flawed if a dubious assumption is required to achieve the goal. In addition, various forms of redundancy can also be identified by verifying that the goal is achieved even after simplifying the protocol. This simplification usually involves eliminating double encryption and redundant assumptions. The next chapter discusses the mechanization of protocol verification in BAN logic.

Chapter 3

Machine-aided BAN Analysis

This chapter describes a protocol analyzer program, written in Prolog, which automates BAN logic analysis of protocols. Machine-aided analysis of the Needham-Schroeder shared-key protocol carried out by the protocol analyzer illustrates how this program aids in identifying flaws and redundancies in protocols.

3.1 Introduction

In automating BAN logic, the readiness of this formalism for mechanization needs to be considered. The first step in analyzing a protocol is idealization — the process of transforming the informal notation describing the protocol into logical formulae. A parser can be written to perform this task if precise rules to carry out idealization exist. However, no such rules exist for BAN logic, and idealization is only possible after the protocol is intuitively understood. Thus idealization has to be carried out manually by following the guidelines mentioned in Section 2.2.1.

Once idealization is done, initial assumptions are made and inference rules are applied to derive the statements held by the principals as a consequence of the message exchange. During analysis, the process of deriving statements by applying the inference rules may be repeated several times as new assumptions are found to be necessary for the protocol to achieve its intended goal. This process is amenable to mechanization, and a program capable of

inferencing can serve as a useful tool in the analysis.

It is desirable that such a program have the following features and capabilities:

- **Protocol-Independence**

The inference rules built into the program must be independent of any specific protocol; i.e., the inference rules need not be re-written or re-ordered for each protocol to be analyzed.

- **Open-ended Analysis**

The program should not be restricted to verifying whether a particular statement in the logic holds for the input protocol. It should generate the complete set of statements describing the state of the principals at the end of the protocol run. In some cases the same statement can be derived from alternate premises or by applying a different inference rule. The program should be capable of generating all derivations of the inferred statements. This enables identification of redundant assumptions and is also a pre-requisite for probabilistic BAN analysis [4], which is discussed in the next chapter.

- **Proof Explanation**

The program should provide a proof-explanation facility to generate a stepwise description of the inferences leading to a particular statement which holds for the input protocol. This facility is useful to obtain machine-generated proofs of goals achieved by protocols. It can also help identify the possible reasons for the failed proof of a desired goal, by permitting manual inspection of the proof of some weaker conclusion.

A NU-Prolog [17] program to automate BAN logic analysis is proposed in [6]. This program suffers from a number of drawbacks. Although the inference rules built into the program are not specific to a particular protocol, the program is not protocol-independent. The program is sensitive to the order of clauses defining the inference rules and can go into “infinite loops”

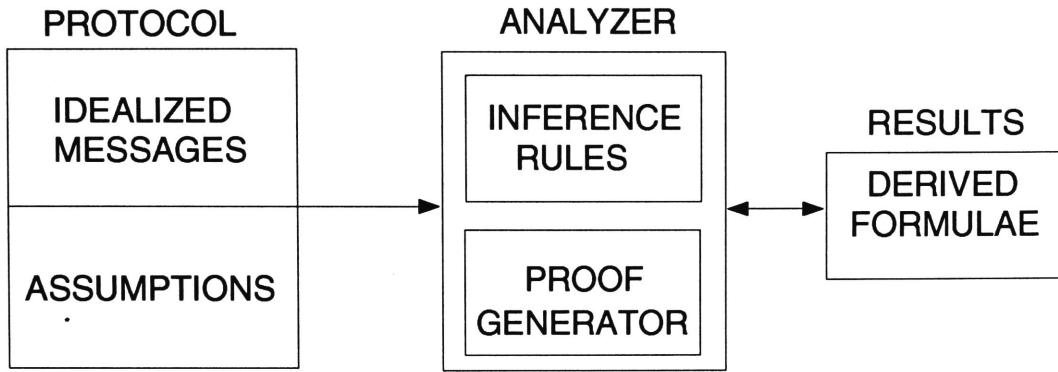


Figure 3.1: Protocol Analysis.

while analyzing different protocols. There is no way of knowing beforehand the correct clause order which prevents an infinite loop while analyzing a given protocol. As a result *ad hoc* manual re-ordering of the program clauses is required. Also, the program does not provide open-ended analysis, as the search path for satisfying a goal is closed when only one solution is found. Further, the program lacks a proof explanation facility. One more Prolog program for mechanizing BAN logic analysis is proposed in [5]. The main limitation of this program is that it is protocol-specific and requires re-writing of the program clauses for each protocol to be analyzed.

The next section outlines the strategy used in a protocol analyzer program which overcomes the limitations of the programs proposed in [6, 5].

3.2 The Protocol Analyzer

Figure [3.1] illustrates the approach used in the program we have developed. The *analyzer module* is the protocol-independent component of the system. It takes as input a protocol specification in the form of idealized messages and assumptions. The analyzer uses a forward-chaining control strategy to generate all statements that can be derived by applying the inference rules to the protocol specification, and stores these statements in a database. It repeatedly cycles through the inference rules and the statements stored in the database until no more statements can be added to the database. A statement is added either if it has not been derived earlier or a new derivation

of it is found. The latter occurs when a statement can be derived from a different set of premises, or by an application of a different inference rule, than used in earlier derivations. Forward-chaining terminates upon exhausting all derivations of the statements attained by the principals executing the protocol. The set of statements describing the state of the principals at the end of the protocol can then be examined by inspecting the database. A proof of a statement that holds for the protocol can be constructed by following the inference steps stored in the database leading to this statement.

The Prolog program developed using the approach given above is discussed in the next section.

3.3 The Prolog Program

The representation of the BAN logic constructs in our program, and the main predicates defined in the program, are described below. The complete program is given in Appendix C.

3.3.1 Structures

Each BAN logic formula is represented by a Prolog structure. The following structures are used to represent formulae of the logic:

Prolog Structure	BAN Formula
<code>believes(P, X)</code>	$P \models X$
<code>sees(P, X)</code>	$P \triangleleft X$
<code>oncesaid(P, X)</code>	$P \vdash X$
<code>jurisdiction(P, X)</code>	$P \Rightarrow X$
<code>fresh(X)</code>	$\sharp(X)$
<code>goodkey(K, P, Q)</code>	$P \stackrel{K}{\leftrightarrow} Q$
<code>public((K, P)</code>	$\stackrel{K}{\mapsto} P$
<code>secret(X, P, Q)</code>	$P \stackrel{X}{\rightleftharpoons} Q$
<code>encrypt(K, X)</code>	$\{X\}_K$
<code>combined(Y, X)</code>	$\langle X \rangle_Y$

In addition the structure **sends**(P, Q, F) is used to represent the step $P \rightarrow Q : F$, where F is the formula denoting the idealized message. The private key K^{-1} is represented by the structure **inv**(K). Conjunctions are represented as lists of the structures defined above. Principals, keys and nonces are denoted by constant symbols.

3.3.2 Predicates

The function of the various predicates defined in the Prolog program is discussed below. In what follows, the notation P/N denotes the predicate named P with arity N .

Representing Statements

The predicate **fact**/3 is used to represent a statement and its derivation. It takes the form:

```
fact(Index, Formula, reason(PremIs, Rule))
```

where **Formula** is bound to a structure representing a statement. The integer argument **Index** is used to index instances of **fact**/3 in the database. The third argument stores information about the derivation of **Formula**. Specifically, **PremIs** is a list containing the indices of the facts representing the statements used as premises in the derivation of **Formula** by an application of inference rule **Rule**. The list **PremIs** is empty if **Formula** is either an assumption or a protocol message, and **Rule** is then 'Assumption' or 'Step'.

Forward-chaining

The top-level predicate is **analyze**/1 and takes as its argument the name of the file containing facts defining the idealized messages and assumptions of the protocol to be analyzed. It does initialization and starts forward-chaining. During initialization the facts defining the protocol are loaded into the program database and the effect of executing each protocol step

$P \rightarrow Q : F$ is recorded by asserting the fact representing $Q \triangleleft F$. Forward-chaining is performed by the predicate `forward/1`:

```
% forward(Cycle) - Main Driver
% Repeatedly applies the inference rules of the logic. It
% succeeds when no new statements can be derived. The
% argument Cycle is incremented with every rule cycle
% performed.
forward(Cycle) :-
    Cycle > 0,
    done,
    write('Analyzed in '), write(Cycle), write(' cycles'), nl.

forward(Cycle) :-
    apply_rules,
    NextCycle is Cycle + 1,
    forward(NextCycle).
```

The predicate `apply_rules/0` applies the inference rules of the logic by invoking various two-arity predicates named after them. A call to a predicate defining an inference rule binds its first argument to the conclusion entailed by the inference rule and binds its second argument to the derivation information of this conclusion. A failure-driven loop forces all inference rules to be applied to the facts stored in the database.

```
% apply_rules - Applies all inference rules
% Applies inference rules to the facts in the database
% and stores new formulae derived. Always succeeds.
apply_rules :-
    (
        sight(Conclusion, Reason)
    ;
        message_decryption(Conclusion, Reason)
    )
```

```

        message_meaning(Conclusion, Reason)
    ;
        freshness(Conclusion, Reason)
    ;
        nonce_verification(Conclusion, Reason)
    ;
        belief(Conclusion, Reason)
    ;
        utterance(Conclusion, Reason)
    ;
        jurisdiction(Conclusion, Reason)
    ;
        bidirectionality(Conclusion, Reason)
),
addfact(Conclusion, Reason),
fail.
apply_rules.

```

The inference rules belonging to the same group are defined as separate clauses of the predicate defining them. For example, the belief rules are defined as separate clauses of the predicate `belief/2`:

```

% Belief rules
belief(believes(P, M), reason([I], 'Belief')) :-
    fact(I, believes(P, L), _),
    member(M, L).
belief(believes(P, believes(Q, M)), reason([I], 'Belief')) :-
    fact(I, believes(P, believes(Q, L)), _),
    member(M, L).

```

The last clause of `apply_rules/0` is a fact `apply_rules`. Thus the goal `apply_rules` always succeeds at the end of the current cycle. A test is made at the beginning of every cycle to determine whether any statements were added during the previous cycle. The predicate `done/0` is used for this purpose. If no statements have been added to the program database in the

previous cycle, then `done` succeeds and forward-chaining terminates. Otherwise, the cycle number is updated and the next cycle is invoked recursively.

Proof Explanation

The predicate `explain_proof/1` provides a stepwise description of proofs of statements which hold for the input protocol. The goal `explain_proof(Goal)` invokes the predicate `build_proof/2` to build a list `Proof` containing the indices of the database facts which were used in a derivation of `Goal`. This list is then passed to the predicate `write_proof/1` which prints the formatted proof. A failure-driven loop forces `explain_proof(Goal)` to output all proofs of `Goal`:

```
% explain_proof(Goal) - Proof Explanation
% Explains all proofs of statement Goal.
explain_proof(Goal) :-
    build_proof(Goal, Proof),
    write_proof(Proof),
    nl,
    fail.
explain_proof(_).

% build_proof(Goal, Proof) - Builds index list of proofs
% The list Proof represents a minimal proof of statement Goal.
build_proof(Goal, Proof) :-
    fact(Index, Goal, _),
    proof(Index, ProofDup),
    remove_dups([Index|ProofDup], RevProof),
    reverse(RevProof, Proof).
```

The goal `build_proof(Goal, Proof)` invokes the predicate `proof/2` which traverses the database facts used in a derivation of `Goal` and returns their indices in the list `ProofDup`. The list `ProofDup` is further processed to eliminate duplicate indices in it which can occur when the same fact is used as

a premise in more than one inference step in the derivation. The list without duplicates, `RevProof`, is finally reversed to obtain the list `Proof`. The list `Proof` represents a minimal proof of `Goal` because only the database facts leading to a derivation of `Goal` are traversed while constructing this list. Also, since the database facts are traversed in the order they are linked together, this list is one representative member of an equivalence class of minimal proofs of `Goal`. The predicate `proof/2` is defined as:

```
% proof(Index, ProofDup) - Traverses database
% Traverses the database to build the list ProofDup containing
% the indices of facts used in a derivation of the fact with
% index Index.
proof(Index, ProofDup) :-
    fact(Index, _, reason(Premises, _)),
    append(Premises, ProofOfPremises, ProofDup),
    prooflist(Premises, ProofOfPremises).

prooflist([], []).
prooflist([Premise|Premises], TotalProof) :-
    proof(Premise, ProofOfPremise),
    prooflist(Premises, ProofOfPremises),
    append(ProofOfPremise, ProofOfPremises, TotalProof).
```

The predicate `write_proof/2` produces a formatted output of the proof from the list `ProofIs`. The proof is output as a sequence of steps leading to the conclusion `G`:

```
% write_proof(Proof) - Prints proof
% Prints formatted proof from the list Proof.
write_proof(Proof) :-
    write_proof(Proof, Proof).

write_proof([], _).
write_proof([Index|Indices], Proof) :-
```

```

fact(Index, Formula, reason(Premises, Rule)),
ith(StepNo, Proof, Index),
write(StepNo), write(' '),
write(' '), write(Formula), tab(2),
write('{'),
write_list(Premises, Proof), write(Rule),
write('}'), nl,
write_proof(Indices, Proof).

```

The steps of the proof are numbered consecutively and each step is printed on a new line. Each step contains either an assumption or a conclusion drawn from statements in earlier lines. The inference rule and the line numbers of the premises used in deriving a statement are listed to the right of each statement.

Avoiding Infinite Loops

In the database there can be several instances of `fact/3` containing the same statement. These multiple facts for the same statement correspond to different derivations of this statement. A check is made before adding the fact representing a derivation to determine whether its insertion in the database would result in an infinite loop. Such a loop can arise either if the inference has already been made earlier or the derived statement is itself used earlier in the derivation of one of the premises being used in the current derivation. It is easy to realize the need for this check by considering repeated application of the bidirectionality inference rule. The predicate `check/2` is used to detect derivations which would lead to infinite loops:

```

% check(Formula, Reason) - Loop checker
% Succeeds if insertion of the fact representing the derivation
% of the statement Formula whose derivation information is in
% Reason would lead to an infinite loop; fails otherwise.

% Inference already made
check(Formula, Reason)

```

```

    fact(_, Formula, Reason),
    !,
    fail.

% Check if the inferred statement Formula was used earlier in
% deriving a premise in PremIs
check(Formula, reason(PremIs, _)) :-
    findall(Index, fact(Index, Formula, _), Indices),
    Indices \= [],
    !,
    checkall(PremIs, Indices).

% New derivation
check(_, _).

checkall([], _).
checkall([PremIndex|PremIndices], Indices) :-
    proof(PremIndex, ProofOfPrem),
    intersection(Indices, [PremIndex|ProofOfPrem], []),
    checkall(PremIndices, Indices).

```

Redundant Assumptions

Redundant assumptions can be detected from the multiple proofs of the desired goal arising out of them. The predicate `goal/1` is used to define the desired goals of various principals. It takes the form `goal(Goal)` where `Goal` is bound to the structure representing a desired goal of a principal.

The top-level predicate for detecting redundant assumptions is `chk_red/3`. The goal `chk_red(P, RedAssumption, MinAssumptions)` succeeds when the list `MinAssumptions` is a minimal set of assumptions of principal `P`, which is required to attain the desired goal of `P`, excluding a redundant assumption `RedAssumption`.

```

% chk_red(P, RedAssumption, MinAssumptions)

```

```

% Identifies redundant assumptions
% The list MinAssumptions is the minimal set of assumptions
% excluding a redundant assumption RedAssumption of principal P.
chk_red(P, RedAssumption, MinAssumptions) :-
    findall(Assumption, is_ass(P, Assumption, _), Assumptions),
    is_ass(P, RedAssumption, Index),
    findall(Goal, (goal(Goal), arg(1, Goal, P)), Goals),
    Goals \= [],
    chk_allfmla(Index, Goals),
    select(RedAssumption, Assumptions, MinAssumptions).

% is_ass(P, Assumption, Index) - Gets assumptions
% Index is the index of the database fact representing
% the assumption Assumption of principal P.
is_ass(P, Assumption, Index) :-
    fact(Index, Assumption, reason([], 'Assumption')),
    arg(1, Assumption, P).

% chk_allfmla(Index, Formulae) - Checks proofs of all formulae
chk_allfmla(_, []).
chk_allfmla(Index, [Formula|Formulae]) :-
    chk_fmla(Index, Formula),
    chk_allfmla(Index, Formulae).

```

The goal `chk_fmla(Index, Formula)` builds the list of minimal proofs, `Proofs`, of a statement `Formula`. It then invokes the predicate `chk_proof/2` to check whether the database fact with index `Index` is redundant for attaining `Formula`:

```

% chk_fmla(Index, Formula) - Checks all proofs of a formula
chk_fmla(Index, Formula) :-
    findall(Proof, build_proof(Formula, Proof), Proofs),
    chk_proof(Index, Proofs).

```

The goal `chk_proof(Index, Proofs)` succeeds when a proof whose index set does not contain `Index` is found in the list of proofs `Proofs`:

```
chk_proof(Index, [Proof|_]) :-
    not(member(Index, Proof)),
    !.
chk_proof(Index, [_|Proofs]) :-
    chk_proof(Index, Proofs).
```

3.4 The Needham-Schroeder Shared-Key Protocol

The application of the protocol analyzer program as a tool in the BAN analysis of protocols is now illustrated by means of an example. In Chapter 2 it was shown how BAN logic analysis of the Needham-Schroeder shared-key protocol exposes the flaw and redundancies in the protocol. The use of the protocol analyzer program as an aid in the BAN logic analysis of this protocol is demonstrated below.

3.4.1 Program Representation

Each idealized message and assumption described in Section 2.5 is defined as a separate instance of `fact/3`. The following set of facts comprising the protocol specification are stored in a file named “nssk”:

```
% Idealized protocol
fact(1, sends(s, a, encrypt(kas, [na, goodkey(kab, a, b), encrypt(kbs, goodkey(kab, a, b))])),
    reason([], 'Step')).
fact(2, sends(a, b, encrypt(kbs, goodkey(kab, a, b))), reason([], 'Step')).
fact(3, sends(b, a, encrypt(kab, [nb, goodkey(kab, a, b)])), reason([], 'Step')).
fact(4, sends(a, b, encrypt(kab, [nb, goodkey(kab, a, b)])), reason([], 'Step')).

% Assumptions
fact(5, believes(a, goodkey(kas, a, s)), reason([], 'Assumption')).
fact(6, believes(a, jurisdiction(s, goodkey(_, a, b))), reason([], 'Assumption')).
fact(7, believes(a, fresh(na)), reason([], 'Assumption')).
fact(8, believes(b, goodkey(kbs, b, s)), reason([], 'Assumption')).
fact(9, believes(b, jurisdiction(s, goodkey(_, a, b))), reason([], 'Assumption')).
```

```

fact(10, believes(b, fresh(nb)), reason([], 'Assumption')).
fact(11, believes(s, goodkey(kbs, b, s)), reason([], 'Assumption')).
fact(12, believes(s, goodkey(kab, a, b)), reason([], 'Assumption')).
fact(13, believes(s, goodkey(kas, a, s)), reason([], 'Assumption')).

```

3.4.2 Open-ended Analysis

Open-ended analysis is performed by invoking the goal `analyze(nssk)`:

```

[ns loaded]
Analyzed in 4 cycles

```

The statements attained by the principals during the execution of the protocol can be obtained by inspecting the database:

```

| ?- listing(fact/3).
fact(1,sends(s,a,encrypt(kas,[na,goodkey(kab,a,b),encrypt(kbs,goodkey(kab,a,b))])),
reason([],Step)).
fact(2,sends(a,b,encrypt(kbs,goodkey(kab,a,b))),reason([],Step)).
fact(3,sends(b,a,encrypt(kab,[nb,goodkey(kab,a,b)])),reason([],Step)).
fact(4,sends(a,b,encrypt(kab,[nb,goodkey(kab,a,b)])),reason([],Step)).
fact(5,believes(a,goodkey(kas,a,s)),reason([],Assumption)).
fact(6,believes(a,jurisdiction(s,goodkey(A,a,b))),reason([],Assumption)).
fact(7,believes(a,fresh(na)),reason([],Assumption)).
fact(8,believes(b,goodkey(kbs,b,s)),reason([],Assumption)).
fact(9,believes(b,jurisdiction(s,goodkey(A,a,b))),reason([],Assumption)).
fact(10,believes(b,fresh(nb)),reason([],Assumption)).
fact(11,believes(s,goodkey(kbs,b,s)),reason([],Assumption)).
fact(12,believes(s,goodkey(kab,a,b)),reason([],Assumption)).
fact(13,believes(s,goodkey(kas,a,s)),reason([],Assumption)).
fact(14,sees(a,encrypt(kas,[na,goodkey(kab,a,b),encrypt(kbs,goodkey(kab,a,b))])),
reason([1],Seeing)).
fact(15,sees(b,encrypt(kbs,goodkey(kab,a,b))),reason([2],Seeing)).
fact(16,sees(a,encrypt(kab,[nb,goodkey(kab,a,b)])),reason([3],Seeing)).
fact(17,sees(b,encrypt(kab,[nb,goodkey(kab,a,b)])),reason([4],Seeing)).
fact(18,believes(a,goodkey(kas,s,a)),reason([5],Bidirectionality)).
fact(19,believes(b,goodkey(kbs,s,b)),reason([8],Bidirectionality)).
fact(20,believes(s,goodkey(kbs,s,b)),reason([11],Bidirectionality)).
fact(21,believes(s,goodkey(kab,b,a)),reason([12],Bidirectionality)).
fact(22,believes(s,goodkey(kas,s,a)),reason([13],Bidirectionality)).
fact(23,sees(a,[na,goodkey(kab,a,b),encrypt(kbs,goodkey(kab,a,b))]),
reason([18,14],Message decryption)).
fact(24,sees(b,goodkey(kab,a,b)),reason([19,15],Message decryption)).
fact(25,believes(a,oncesaid(s,[na,goodkey(kab,a,b),encrypt(kbs,goodkey(kab,a,b))])),
reason([18,14],Message meaning)).
fact(26,believes(b,oncesaid(s,goodkey(kab,a,b))),reason([19,15],Message meaning)).
fact(27,believes(a,fresh([na,goodkey(kab,a,b),encrypt(kbs,goodkey(kab,a,b))])),
reason([7],Freshness)).

```

```

fact(28,believes(a,believes(s,[na,goodkey(kab,a,b),encrypt(kbs,goodkey(kab,a,b))])),
reason([27,25],Nonce verification)).
fact(29,believes(a,believes(s,na)),reason([28],Belief)).
fact(30,believes(a,believes(s,goodkey(kab,a,b))),reason([28],Belief)).
fact(31,believes(a,believes(s,encrypt(kbs,goodkey(kab,a,b)))),reason([28],Belief)).
fact(32,believes(a,oncesaid(s,na)),reason([25],Utterance)).
fact(33,believes(a,oncesaid(s,goodkey(kab,a,b))),reason([25],Utterance)).
fact(34,believes(a,oncesaid(s,encrypt(kbs,goodkey(kab,a,b)))),reason([25],Utterance)).
fact(35,believes(a,goodkey(kab,a,b)),reason([6,30],Jurisdiction)).
fact(36,believes(a,goodkey(kab,b,a)),reason([35],Bidirectionality)).
fact(37,believes(a,believes(s,goodkey(kab,b,a))),reason([30],Bidirectionality)).
fact(38,sees(a,na),reason([23],Sight)).
fact(39,sees(a,goodkey(kab,a,b)),reason([23],Sight)).
fact(40,sees(a,encrypt(kbs,goodkey(kab,a,b))),reason([23],Sight)).
fact(41,sees(a,[nb,goodkey(kab,a,b)]),reason([36,16],Message decryption)).
fact(42,sees(b,[nb,goodkey(kab,a,b)]),reason([26,17],Message decryption)).
fact(43,believes(a,oncesaid(b,[nb,goodkey(kab,a,b)])),reason([36,16],Message meaning)).
fact(44,believes(a,believes(s,na)),reason([7,32],Nonce verification)).
fact(45,believes(a,oncesaid(b,nb)),reason([43],Utterance)).
fact(46,believes(a,oncesaid(b,goodkey(kab,a,b))),reason([43],Utterance)).
fact(47,sees(a,nb),reason([41],Sight)).
fact(48,sees(a,goodkey(kab,a,b)),reason([41],Sight)).
fact(49,sees(b,nb),reason([42],Sight)).
fact(50,sees(b,goodkey(kab,a,b)),reason([42],Sight)).

```

yes

This database can then be queried to determine whether the protocol achieves its intended goal:

```

| ?- fact(_, believes(a, goodkey(kab, a, b)), _).

yes
| ?- fact(_, believes(a, believes(b, goodkey(kab, a, b))), _).

no
| ?- fact(_, believes(b, goodkey(kab, a, b)), _).

no
| ?- fact(_, believes(b, believes(a, goodkey(kab, a, b))), _).

no

```

The results of the above queries show that for the given assumptions only the first-order goal of A is achieved. The proof of $A \models A \xleftrightarrow{K_{ab}} B$ is:

```

| ?- explain_proof(believes(a, goodkey(kab, a, b))).
1. sends(s,a,encrypt(kas,[na,goodkey(kab,a,b),encrypt(kbs,goodkey(kab,a,b))])) {Step}

```



```

2. believes(a,goodkey(kas,a,s)) {Assumption}
3. sees(a,encrypt(kas,[na,goodkey(kab,a,b),encrypt(kbs,goodkey(kab,a,b))])) {1, Seeing}
4. believes(a,goodkey(kas,s,a)) {2, Bidirectionality}
5. believes(a,fresh(na)) {Assumption}
6. believes(a,oncesaid(s,[na,goodkey(kab,a,b),encrypt(kbs,goodkey(kab,a,b))]))
{4, 3, Message meaning}
7. believes(a,fresh([na,goodkey(kab,a,b),encrypt(kbs,goodkey(kab,a,b))])) {5, Freshness}
8. believes(a,believes(s,[na,goodkey(kab,a,b),encrypt(kbs,goodkey(kab,a,b))]))
{7, 6, Nonce verification}
9. believes(a,believes(s,goodkey(kab,a,b))) {8, Belief}
10. believes(a,jurisdiction(s,goodkey(_1259604,a,b))) {Assumption}
11. believes(a,goodkey(kab,a,b)) {10, 9, Jurisdiction}

yes

```

The following observations can be made by inspecting the statements stored in the database. Principal A attains the weaker conclusion $A \models B \vdash A \stackrel{K_{ab}}{\leftrightarrow} B$ (fact no. 46) in place of his second-order goal $A \models B \models A \stackrel{K_{ab}}{\leftrightarrow} B$. Principal B achieves the weak conclusion $B \models S \vdash A \stackrel{K_{ab}}{\leftrightarrow} B$ (fact no. 26) in place of his first-order goal $B \models A \stackrel{K_{ab}}{\leftrightarrow} B$.

The weak conclusion reached by B suggests that B must make the additional assumption $B \models \#(A \stackrel{K_{ab}}{\leftrightarrow} B)$ to attain his first-order goal. Once this dubious assumption is made, B achieves both his goals. The proof of his first-order goal is shown below:

```

| ?- explain_proof(believes(b, goodkey(kab, a, b))).
1. sends(a,b,encrypt(kbs,goodkey(kab,a,b))) {Step}
2. believes(b,goodkey(kbs,b,s)) {Assumption}
3. sees(b,encrypt(kbs,goodkey(kab,a,b))) {1, Seeing}
4. believes(b,goodkey(kbs,s,b)) {2, Bidirectionality}
5. believes(b,oncesaid(s,goodkey(kab,a,b))) {4, 3, Message meaning}
6. believes(b,fresh(goodkey(kab,a,b))) {Assumption}
7. believes(b,believes(s,goodkey(kab,a,b))) {6, 5, Nonce verification}
8. believes(b,jurisdiction(s,goodkey(_1258748,a,b))) {Assumption}
9. believes(b,goodkey(kab,a,b)) {8, 7, Jurisdiction}

yes

```

The four proofs of the second-order goal of B are given in Appendix B.2. By inspecting the weak conclusion reached by A in place of his second-order goal, it is clear that A can attain this goal if the key K_{ab} generated by S is fresh. This requirement is expressed by the additional assumptions $S \models \#(A \stackrel{K_{ab}}{\leftrightarrow} B)$ and $A \models S \Rightarrow \#(A \stackrel{K}{\leftrightarrow} B)$. Also, the idealized Message 2 is

modified to include the additional formula $\sharp(A \stackrel{K_{ab}}{\leftrightarrow} B)$. With these changes made, it can be verified by running the program again that A attains his second-order goal. This also gives rise to an alternate derivation for the first-order goal of A as shown below:

1. `sends(s,a,encrypt(kas,[na,goodkey(kab,a,b),fresh(goodkey(kab,a,b)),
encrypt(kbs,goodkey(kab,a,b))])) {Step}`
2. `believes(a,goodkey(kas,a,s)) {Assumption}`
3. `sees(a,encrypt(kas,[na,goodkey(kab,a,b),fresh(goodkey(kab,a,b)),
encrypt(kbs,goodkey(kab,a,b))])) {1, Seeing}`
4. `believes(a,goodkey(kas,s,a)) {2, Bidirectionality}`
5. `believes(a,oncesaid(s,[na,goodkey(kab,a,b),fresh(goodkey(kab,a,b)),
encrypt(kbs,goodkey(kab,a,b))])) {4, 3, Message meaning}`
6. `believes(a,fresh(na)) {Assumption}`
7. `believes(a,fresh([na,goodkey(kab,a,b),fresh(goodkey(kab,a,b)),
encrypt(kbs,goodkey(kab,a,b))])) {6, Freshness}`
8. `believes(a,believes(s,[na,goodkey(kab,a,b),fresh(goodkey(kab,a,b)),
encrypt(kbs,goodkey(kab,a,b))])) {7, 5, Nonce verification}`
9. `believes(a,believes(s,fresh(goodkey(kab,a,b)))) {8, Belief}`
10. `believes(a,jurisdiction(s,fresh(goodkey(_1261200,a,b)))) {Assumption}`
11. `believes(a,oncesaid(s,goodkey(kab,a,b))) {5, Utterance}`
12. `believes(a,fresh(goodkey(kab,a,b))) {10, 9, Jurisdiction}`
13. `believes(a,believes(s,goodkey(kab,a,b))) {12, 11, Nonce verification}`
14. `believes(a,jurisdiction(s,goodkey(_1261444,a,b))) {Assumption}`
15. `believes(a,goodkey(kab,a,b)) {14, 13, Jurisdiction}`

yes

The alternate proof of $A \models A \stackrel{K_{ab}}{\leftrightarrow} B$ given above arises as follows. Principal A first deduces $\sharp(A \stackrel{K_{ab}}{\leftrightarrow} B)$ as a nonce from Message 2 by using $A \models S \Rightarrow \sharp(A \stackrel{K_{ab}}{\leftrightarrow} B)$. The freshness of this nonce is further used to obtain $A \models S \models A \stackrel{K_{ab}}{\leftrightarrow} B$ from which $A \models A \stackrel{K_{ab}}{\leftrightarrow} B$ follows by applying the jurisdiction rule as before. The three proofs of the second-order goal of A are given in Appendix B.2. The following table gives the number of minimal proofs of the goals of principals A and B :

Goal	No. of Proofs
$A \models A \stackrel{K_{ab}}{\leftrightarrow} B$	2
$B \models A \stackrel{K_{ab}}{\leftrightarrow} B$	1
$A \models B \models A \stackrel{K_{ab}}{\leftrightarrow} B$	4
$B \models A \models A \stackrel{K_{ab}}{\leftrightarrow} B$	3

3.4.3 Redundant Assumptions

The goal of the protocol is represented by defining a separate fact of `goal/1` for each formula to be attained, as follows.

```
| ?- [user].
goal(believes(a, goodkey(kab, a, b))).
goal(believes(a, believes(b, goodkey(kab, a, b)))).
goal(believes(b, goodkey(kab, a, b))).
goal(believes(b, believes(a, goodkey(kab, a, b)))).
[user compiled, cpu time used: 0.29102 seconds]
[user loaded]
```

yes

The assumptions made by principal *A* are:

```
| ?- is_ass(a, A, _).

A = believes(a, goodkey(kas, a, s));
A = believes(a, jurisdiction(s, goodkey(_1257304, a, b)));
A = believes(a, jurisdiction(s, fresh(goodkey(_1257312, a, b))));
A = believes(a, fresh(na));
```

no

The following query shows that no assumption made by *A* is redundant:

```
| ?- chk_red(a, A, As).
```

no

For principal *B*, the assumptions made are:

```
| ?- is_ass(b, A, _).

A = believes(b, goodkey(kbs, b, s));
A = believes(b, jurisdiction(s, goodkey(_1257304, a, b)));
A = believes(b, fresh(nb));
A = believes(b, fresh(goodkey(kab, a, b)));
```

no

The following query returns the redundant assumption made by *B*:

```
| ?- chk_red(b, A, As).

A = believes(b, fresh(nb))
As = [believes(b, goodkey(kbs, b, s)), believes(b, jurisdiction(s, goodkey(_1257384, a,
b))), believes(b, fresh(goodkey(kab, a, b)))];
```

no

Also, it can be checked that the double encryption in Message 2 is redundant by mechanically verifying that the protocol achieves its intended goal even if $\{A, K_{ab}\}_{K_{bs}}$ is not encrypted with K_{as} in this message.

3.5 Remarks

The protocol analyzer program described in Section 3.3 serves as a useful tool in the BAN logic analysis of protocols. Machine-aided BAN analysis of several other well-known protocols using this program is illustrated in Appendix B. The program is protocol-independent and generates all derivations of the complete set of formulae attained by the principals executing a given protocol. The proof explanation facility built into the program provides machine-generated proofs of any formula attained. In addition, the program enables automatic identification of redundant assumptions. The next chapter outlines the probabilistic BAN logic analysis proposed by Campbell, Safavi-Naini and Pleasants in [4].

Chapter 4

Probabilistic BAN Analysis

This chapter discusses the application of probabilistic logic to BAN analysis of protocols. The use of the protocol analyzer program as an aid in carrying out probabilistic BAN analysis [4] is illustrated by means of the Needham-Schroeder shared-key protocol. Finally, the misinterpretation in [3] of the significance of the probabilistic analysis of this protocol carried out in [4] is resolved.

4.1 Introduction

During protocol analysis, a conclusion derived may be implied by different assumptions or inference rules. For the purpose of determining whether a particular goal is achieved or not for the given set of assumptions, it suffices to ignore alternate derivations of the goal. However, the protocol analyzer generates all derivations of the inferred statements. This facilitates probabilistic analysis, as discussed in the next section.

4.2 Probabilistic Logic

In the BAN model of reasoning about authentication protocols, principals are assumed to be totally committed to the logical statements describing their state. Specifically, the beliefs established during a protocol run are always considered to be true. In a hostile environment, however it may not

be appropriate for the principals to assent fully to the initial assumptions and the inference rules of the logic. A probabilistic extension of BAN logic proposed by Campbell et al. [4] models insecurity by attaching probabilities to the statements and inference rules of the logic. The probability of the desired goal then gives a measure of the trust that can be put in the goal.

The problem of determining the probability of a derived formula is shown in [7] to reduce to a linear programming problem and this reduction is outlined below.

4.2.1 Probability of Derived Formula

Let $B = \{b_1, \dots, b_m\}$ be the set of assumptions and rule instances used in deriving a conclusion c . A *possible world* for the set B is any binary truth-value assignment to the elements of B . Therefore, the number of possible worlds for B is 2^m . Further, let p_i be the probability of b_i for $1 \leq i \leq m$ and let π_j be the probability of the possible world W_j for $1 \leq j \leq 2^m$. The individual π_j sum to 1 since the possible worlds are mutually exclusive and exhaustive. In the model proposed in [4], the probability p_i of b_i is taken to be the sum of the probabilities of the possible worlds in which b_i is assigned a truth-value of 1. This can be rephrased mathematically by introducing matrix notation, as follows.

Let the probabilities p_i be arranged in a m -dimensional column matrix $P = (p_1, \dots, p_m)$. Similarly, let the probabilities π_j be arranged in a 2^m -dimensional column matrix $\pi = (\pi_1, \dots, \pi_{2^m})$. The element in the j th row of π is the probability of the j th possible world W_j . Let W be the $m \times 2^m$ matrix (w_{ij}) such that w_{ij} is the truth-value assigned to b_i in the possible world W_j . Then, the probabilities of the assumptions and rule instances in B are related to the probabilities of the possible worlds by the matrix equation:

$$P = W\pi \quad (4.1)$$

Also,

$$\sum_{j=1}^{2^m} \pi_j = 1 \quad (4.2)$$

The probability of the conclusion c is given by:

$$p(c) = q\pi \quad (4.3)$$

where $q = (q_j)$ is the 2^m -dimensional 0 – 1 row matrix such that $q_j = 1$ if the b_i 's which are assigned truth-value 1 in the possible world W_j , form a minimal proof of c .

To determine $p(c)$, one therefore needs to compute the matrix π by solving the set of simultaneous linear equations (4.1)–(4.2). In the general case, equation (4.1) is underdetermined and permits many solutions for π , and hence one can only obtain bounds on the probability $p(c)$. A lower bound L on $p(c)$ can be obtained by solving the following linear programming problem:

$$L = \min(q\pi)$$

subject to the constraints:

$$\begin{aligned} W\pi &= P \\ \sum_{j=1}^{2^m} \pi_j &= 1 \end{aligned}$$

From the viewpoint of protocol security, the lower bound L , is significant since it is a measure of the minimum trust that can be put in the conclusion c . For the purpose of probabilistic analysis, it suffices to have one minimal proof from each equivalence class. As mentioned earlier in Section 3.3.2, the proof generator in the inference engine of our protocol analyzer program constructs one representative of each equivalence class of minimal proofs of the inferred statements for the given set of assumptions.

4.2.2 Protocol Analysis

Probabilistic analysis of protocols as proposed by Campbell et al. [4, 5] involves calculating the minimum trust that can be placed in the goal of the protocol. It is carried out by determining all minimal proofs of the goal and then attaching probabilities to assumptions and rule instances used in the proofs.

The operational meaning of attaching probabilities to statements and inference rules of the logic is given below, following [4].

- $P \models P \overset{K}{\leftrightarrow} Q$

The probability assigned to this statement reflects principal P 's confidence in the ability of P and Q to protect the key.

- $P \models \sharp(X)$

This statement is assigned a high probability when P can accept messages containing X as fresh.

- $P \models Q \vdash X$

The probability assigned to this statement refers to P 's confidence that no substitution has taken place in the message X sent by Q .

- $P \models S \Rightarrow P \overset{K}{\leftrightarrow} Q$

The probability assigned to this statement reflects P 's confidence in the competence of S in generating good session keys.

- $P \triangleleft X$

The interpretation of this statement is that P sees the message X , and the statement is always assigned probability of 1.

The probability attached to an instance of an inference rule reflects the confidence of the principal applying the inference rule that the particular inference holds:

- **Message-meaning rule**

$$\frac{P \models Q \overset{K}{\leftrightarrow} P, P \triangleleft \{X\}_K}{P \models Q \vdash X}$$

The probability attached to an instance of this rule reflects the physical security of the communication channel between P and Q , and the resistance of the encryption algorithm used to encrypt X .

- **Nonce-verification rule**

$$\frac{P \models \#(X), P \models Q \vdash X}{P \models Q \models X}$$

The probability attached to an instance of this rule reflects the degree to which P trusts Q on the truth of X .

Usually, instances of all inference rules except the message-meaning and nonce-verification inference rules are assumed to be certain, and are assigned a probability of 1.

In probabilistic analysis, the problem is of determining the lower bound on the probability of the goal of the protocol knowing the probabilities of various assumptions and rule instances used in the minimal proofs of the goal. In principle, this lower bound can always be determined by solving a linear programming problem. However, the following two theorems given in [4] show that when there are at most two minimal proofs of a conclusion, the lower bound on the probability of the conclusion can be directly expressed in terms of the probabilities of the assumptions and rule instances used in the minimal proofs.

Let $B = \{b_1, \dots, b_m\}$ be the set of assumptions and rule instances used in deriving a conclusion c , and let p_i be the probability of b_i for $1 \leq i \leq m$. Further, let $J \subseteq \{1, \dots, m\}$. Define

$$l(J) = \sum_{i \in J} p_i - |J| + 1,$$

where $|J|$ denotes the number of elements in J .

Theorem 4.1 *Let there exist exactly one minimal proof of the conclusion c from $B = \{b_1, \dots, b_m\}$, and let J be the index set of this minimal proof. Then*

$$P(c) \geq \max(0, l(J)).$$

Theorem 4.2 *Let there exist exactly two minimal proofs of the conclusion c from $B = \{b_1, \dots, b_m\}$, and let J and K be the index sets of the two minimal proofs. Then*

$$P(c) \geq \max(0, l(J), l(K)).$$

The above two theorems are proved in [7]. Moreover, the bounds given above are tight since in each case there is no other lower bound which is less than the one given above. In the general case, involving three or more minimal proofs the problem of obtaining a functional representation for the lower bound remains unsolved [4] and therefore an instance of a linear programming problem needs to be solved to obtain this bound.

4.3 The Needham-Schroeder Shared-Key Protocol

Machine-aided BAN analysis of the Needham-Schroeder Shared-Key protocol was presented in Section 3.4. The use of the protocol analyzer in carrying out probabilistic analysis of this protocol follows.

The idealized messages and the assumptions of this protocol are given below.

Idealized Protocol:

- Message 2 $S \rightarrow A : \{N_a, A \stackrel{K_{as}}{\leftrightarrow} B, \#(A \stackrel{K_{as}}{\leftrightarrow} B), \{A \stackrel{K_{as}}{\leftrightarrow} B\}_{K_{bs}}\}_{K_{as}}$
 Message 3 $A \rightarrow B : \{A \stackrel{K_{as}}{\leftrightarrow} B\}_{K_{bs}}$
 Message 4 $B \rightarrow A : \{N_b, A \stackrel{K_{ab}}{\leftrightarrow} B\}_{K_{ab}}$
 Message 5 $A \rightarrow B : \{N_b, A \stackrel{K_{ab}}{\leftrightarrow} B\}_{K_{ab}}$

Assumptions:

$$\begin{array}{ll}
 A \models A \stackrel{K_{as}}{\leftrightarrow} S & B \models B \stackrel{K_{bs}}{\leftrightarrow} S \\
 S \models A \stackrel{K_{as}}{\leftrightarrow} S & S \models B \stackrel{K_{bs}}{\leftrightarrow} S \\
 S \models A \stackrel{K_{as}}{\leftrightarrow} B & S \models \#(A \stackrel{K_{as}}{\leftrightarrow} B) \\
 A \models S \Rightarrow A \stackrel{K}{\leftrightarrow} B & B \models S \Rightarrow A \stackrel{K}{\leftrightarrow} B \\
 A \models S \Rightarrow \#(A \stackrel{K}{\leftrightarrow} B) & B \models \#(A \stackrel{K_{ab}}{\leftrightarrow} B) \\
 A \models \#(N_a) & B \models \#(N_b)
 \end{array}$$

As noted earlier in Section 3.4, the formula $\#(A \stackrel{K_{as}}{\leftrightarrow} B)$ in the idealized Message 2 and the assumptions $A \models S \Rightarrow \#(A \stackrel{K_{as}}{\leftrightarrow} B)$ and $A \models S \Rightarrow \#(A \stackrel{K_{as}}{\leftrightarrow} B)$ are required for A to attain his second-order goal. Also, B needs to make

the dubious assumption $B \models \#(A \stackrel{K_a}{\leftrightarrow} B)$ to attain his goal. The fact/3 facts defining the idealized protocol and the assumptions given above are stored in the file “nssk”.

4.3.1 Minimal Proofs

The proofs of the goals attained by the principals are obtained by using the proof explanation facility. The only minimal proof of the first-order goal of B is:

```
| ?- explain_proof(believes(b, goodkey(kab, a, b))).
1. sends(a,b,encrypt(kbs,goodkey(kab,a,b))) {Step}
2. believes(b,goodkey(kbs,b,s)) {Assumption}
3. sees(b,encrypt(kbs,goodkey(kab,a,b))) {1, Seeing}
4. believes(b,goodkey(kbs,s,b)) {2, Bidirectionality}
5. believes(b,oncesaid(s,goodkey(kab,a,b))) {4, 3, Message meaning}
6. believes(b,fresh(goodkey(kab,a,b))) {Assumption}
7. believes(b,believes(s,goodkey(kab,a,b))) {6, 5, Nonce verification}
8. believes(b,jurisdiction(s,goodkey(_1258748,a,b))) {Assumption}
9. believes(b,goodkey(kab,a,b)) {8, 7, Jurisdiction}

yes
```

If the redundant assumption $B \models \#(N_b)$ is made, then the second-order goal of B has three minimal proofs which are given in Appendix B.2. Discarding this assumption results in two minimal proofs of this goal:

```
1. sends(a,b,encrypt(kab,[nb,goodkey(kab,a,b)])) {Step}
2. sends(a,b,encrypt(kbs,goodkey(kab,a,b))) {Step}
3. believes(b,goodkey(kbs,b,s)) {Assumption}
4. sees(b,encrypt(kbs,goodkey(kab,a,b))) {2, Seeing}
5. believes(b,goodkey(kbs,s,b)) {3, Bidirectionality}
6. believes(b,oncesaid(s,goodkey(kab,a,b))) {5, 4, Message meaning}
7. believes(b,fresh(goodkey(kab,a,b))) {Assumption}
8. believes(b,believes(s,goodkey(kab,a,b))) {7, 6, Nonce verification}
9. believes(b,jurisdiction(s,goodkey(_1259896,a,b))) {Assumption}
10. sees(b,encrypt(kab,[nb,goodkey(kab,a,b)])) {1, Seeing}
11. believes(b,goodkey(kab,a,b)) {9, 8, Jurisdiction}
12. believes(b,oncesaid(a,[nb,goodkey(kab,a,b)])) {11, 10, Message meaning}
13. believes(b,fresh([nb,goodkey(kab,a,b)])) {7, Freshness}
14. believes(b,believes(a,[nb,goodkey(kab,a,b)])) {13, 12, Nonce verification}
15. believes(b,believes(a,goodkey(kab,a,b))) {14, Belief}

1. sends(a,b,encrypt(kab,[nb,goodkey(kab,a,b)])) {Step}
2. sends(a,b,encrypt(kbs,goodkey(kab,a,b))) {Step}
```

3. `believes(b,goodkey(kbs,b,s))` {Assumption}
4. `sees(b,encrypt(kbs,goodkey(kab,a,b)))` {2, Seeing}
5. `believes(b,goodkey(kbs,s,b))` {3, Bidirectionality}
6. `believes(b,oncesaid(s,goodkey(kab,a,b)))` {5, 4, Message meaning}
7. `believes(b,fresh(goodkey(kab,a,b)))` {Assumption}
8. `believes(b,believes(s,goodkey(kab,a,b)))` {7, 6, Nonce verification}
9. `believes(b,jurisdiction(s,goodkey(_1259764,a,b)))` {Assumption}
10. `sees(b,encrypt(kab,[nb,goodkey(kab,a,b)]))` {1, Seeing}
11. `believes(b,goodkey(kab,a,b))` {9, 8, Jurisdiction}
12. `believes(b,oncesaid(a,[nb,goodkey(kab,a,b)]))` {11, 10, Message meaning}
13. `believes(b,oncesaid(a,goodkey(kab,a,b)))` {12, Utterance}
14. `believes(b,believes(a,goodkey(kab,a,b)))` {7, 13, Nonce verification}

yes

It can be seen from the above output that both the proofs are identical till Step no.12 in which the belief $B \models A \vdash (N_b, A \stackrel{K_{ab}}{\leftrightarrow} B)$ is established. In the first minimal proof, B applies the nonce-verification rule without splitting $(N_b, A \stackrel{K_{ab}}{\leftrightarrow} B)$ to obtain the composite belief $B \models A \models (N_b, A \stackrel{K_{ab}}{\leftrightarrow} B)$. This composite belief is then finally split to obtain the desired second-order goal. On the other hand, B can also first split $(N_b, A \stackrel{K_{ab}}{\leftrightarrow} B)$ to obtain $B \models A \vdash A \stackrel{K_{ab}}{\leftrightarrow} B$ and then finally use the nonce-verification rule to derive his second-order goal. This alternative corresponds to the second minimal proof. Note that in both these proofs, B uses the dubious assumption while applying the nonce-verification rule.

4.3.2 Lower Bounds

Once all the minimal proofs of the goals attained by A and B are ascertained, probabilistic analysis can be carried out. The derivation of the lower bounds on the probabilities of the goals attained by principal B is given below.

Let c_1 and c_2 denote the first-order and second-order goals of principal B respectively:

$$\begin{aligned} c_1 : B &\models A \stackrel{K_{ab}}{\leftrightarrow} B \\ c_2 : B &\models A \models A \stackrel{K_{ab}}{\leftrightarrow} B \end{aligned}$$

By inspecting the minimal proofs of the goals of principal B , it can be seen that the following statements and rule instances are used in the derivation of c_1 and c_2 .

Assumptions:

$$\begin{aligned} a_1 : B &\models B \xleftrightarrow{K_{bs}} S \\ a_2 : B &\models \#(A \xleftrightarrow{K_{ab}} B) \\ a_3 : B &\models S \Rightarrow A \xleftrightarrow{K} B \end{aligned}$$

The redundant assumption $a_4 : B \models \#(N_b)$ is discarded.

Messages:

$$\begin{aligned} m_1 : B &\triangleleft \{A \xleftrightarrow{K_{ab}} B\}_{K_{bs}} \\ m_2 : B &\triangleleft \{N_b, A \xleftrightarrow{K_{ab}} B\}_{K_{ab}} \end{aligned}$$

Rule instances:

$$\begin{aligned} r_1 : & \frac{B \models S \xleftrightarrow{K_{bs}} B, B \triangleleft \{X\}_{K_{bs}}}{B \models S \sim X} \\ r_2 : & \frac{B \models \#(X), B \models S \sim X}{B \models S \models X} \\ r_3 : & \frac{B \models S \Rightarrow X, B \models S \models X}{B \models X} \\ r_4 : & \frac{B \models A \xleftrightarrow{K_{ab}} B, B \triangleleft \{X\}_{K_{ab}}}{B \models A \sim X} \\ r_5 : & \frac{B \models \#(X), B \models A \sim X}{B \models A \models X} \\ r_6 : & \frac{B \models A \models (X, Y)}{B \models A \models Y} \\ r_7 : & \frac{B \models A \sim (X, Y)}{B \models A \sim Y} \\ r_8 : & \frac{B \models \#(X)}{B \models \#(X, Y)} \\ r_9 : & \frac{B \models B \xleftrightarrow{K_{bs}} S}{B \models S \xleftrightarrow{K_{bs}} B} \end{aligned}$$

Let p_1, p_2, p_3 denote the probabilities attached to the assumptions a_1, a_2, a_3 . Both m_1 and m_2 are assigned a probability of 1. The rule instances r_3, r_6, r_7, r_8, r_9 are certain, i.e., have probability 1. Let the rule instances r_1, r_2, r_4, r_5 be assigned probabilities p_5, p_6, p_7, p_8 .

The set of logical statements and rule instances used in the only minimal proof of c_1 is $\{m_1, a_1, a_2, a_3, r_1, r_2, r_3, r_9\}$. Since there exists only one minimal

proof of c_1 , the lower bound on $p(c_1)$ is obtained by applying Theorem 4.1, as:

$$\begin{aligned} p(c_1) &\geq \max(0, 1 + p_1 + p_2 + p_3 + p_5 + p_6 + 1 + 1 - 8 + 1) \\ &\geq \max(0, p_1 + p_2 + p_3 + p_5 + p_6 - 4) \end{aligned}$$

For c_2 , the two sets of assumptions and rule instances used in the minimal proofs are:

- $\{m_1, m_2, a_1, a_2, a_3, r_1, r_2, r_3, r_4, r_5, r_6, r_8, r_9\}$
- $\{m_1, m_2, a_1, a_2, a_3, r_1, r_2, r_3, r_4, r_5, r_7, r_9\}$

By applying Theorem 4.2, the lower bound on $p(c_2)$ is obtained as:

$$\begin{aligned} p(c_2) &\geq \\ &\max(0, 1 + 1 + p_1 + p_2 + p_3 + p_5 + p_6 + 1 + p_7 + p_8 + 1 + 1 + 1 - 13 + 1, \\ &\quad 1 + 1 + p_1 + p_2 + p_3 + p_5 + p_6 + 1 + p_7 + p_8 + 1 + 1 - 12 + 1) \\ &\geq \max(0, p_1 + p_2 + p_3 + p_5 + p_6 + p_7 + p_8 - 6) \end{aligned}$$

From the lower bounds on $p(c_1)$ and $p(c_2)$ obtained above, it can be seen that each of the p_i 's occurring in the expression for a bound play an equal role in determining the value of the bound. Recall that each such p_i is the probability of an assumption or an inference rule used in deriving the goals of principal B . Also, each $p_i \in [0, 1]$.

4.3.3 Interpreting the Bounds

In [4] it is stated that:

If we consider the probabilities of the conclusions above we see that $p(c_1)$ and $p(c_2) \rightarrow 0$ as $p_2 \rightarrow 0$, i.e., the lower bound on the probability of the conclusion tends to zero, as the trust in a_2 tends to zero."

The significance of this result has been seriously misinterpreted in [3], where it is claimed on the basis of the above result that:

The weakness of the Needham and Schroeder protocol was discovered by the extensions of Campbell et al., as the original BAN logic does without using prior knowledge of it.

While it is certainly true that BAN logic analysis is successful in detecting the flaw in this protocol without prior knowledge of it, it is misleading to attribute the same virtue to the probabilistic analysis of Campbell et al. Although, $p(c_1)$ and $p(c_2) \rightarrow 0$ as $p_2 \rightarrow 0$, so do they even when any other $p_i \rightarrow 0$. This cannot imply that the corresponding a_i is responsible for the weakness in the protocol.

Chapter 5

Conclusions

The main objective of this project was to develop an implementation of the BAN logic, and to demonstrate its use in carrying out analysis of authentication protocols. A Prolog program for this task has been developed, and has been applied to several well-known protocols as illustrated in Appendix B. The program generates all derivations of the formulae attained by the principals executing the protocol. This allows identification of redundant assumptions and also facilitates probabilistic analysis of protocols.

The implementation of the inference rules of the logic in the program is independent of any specific protocol. The protocol-independent inference engine enhances its applicability as a general purpose tool, unlike the programs proposed in [5, 6]. The program has proved effective in mechanically verifying the analysis of several well-known protocols, published in [1]. It helped detect a missing assumption in the analysis of the Kerberos protocol [B.3] and pinpoint a mistake in the idealization of the Needham-Schroeder public-key protocol [B.5], given in [1].

The applicability of the program in verifying protocols is determined by the scope of BAN logic [1] itself, which is briefly discussed below.

5.1 Scope

The BAN logic does not attempt to capture the problems arising out of inappropriate uses of cryptosystems. For example, there is no rule for checking

whether a particular key is long enough or not. Although the logic allows for the possibility of hostile intruders, it assumes that the principals executing the protocol are trustworthy. In particular, it does not deal with unauthorized release of secrets. Also, it does not attempt to capture weaknesses of encryption schemes. Specifically, the following assumptions about encryption underlie the rules of the logic:

- Encryption is perfect; i.e., it is computationally infeasible to decipher an encrypted message without knowing the appropriate key(s) and to determine these key(s) from a combination of ciphertext and plaintext. Further, each encrypted unit is integral. For example, it is assumed that it is infeasible to fabricate $\{S, T\}_K$ from $\{S\}_K$, $\{T\}_K$, $\{S, U\}_K$, and $\{V, T\}_K$, without knowing K .
- A principal deciphering an encrypted message would be able to determine whether the right decryption key has been used to recover the message. This requirement is based on the assumption that the plaintext recovered may not be completely unpredictable to the recipient.
- A principal can recognize his own messages. In practice, this is achieved by including direction bits in messages, to identify the sender and the intended recipient of a message.

Moreover, the logic does not distinguish between the different purposes underlying the use of encryption in protocols. It does not capture distinctions arising out of the use of encryption to preserve secrecy, to maintain integrity of a message, or to demonstrate the knowledge of a key. Nonetheless, the logic has proved effective in pointing out flaws and redundancies in several protocols [1].

The protocol analyzer program based on the logic serves as a useful tool in the analysis of authentication protocols. It also provides a basic framework for implementing the extensions of BAN logic proposed in [9, 8].

Appendix A

References

1. M. Burrows, M. Abadi, and R. M. Needham. *A Logic of Authentication*, Research Report 39, Digital Systems Research Center, Palo Alto, California, February 1990. A condensed version of this report appeared in ACM Trans. on Computer Systems, 8(1):18–36, February 1990.
2. A. Liebl. *Authentication in Distributed Systems: A Bibliography*, ACM Operating Systems Review, 27(4):31–41, October 1993.
3. A. Rubin, and P. Honeyman. *Formal Methods for the Analysis of Authentication Protocols*, CITI Technical Report 93–7, Center for Information Technology Integration, University of Michigan, Ann Arbor, October 1993.
4. E. Campbell, R. Safavi-Naini, and P. Pleasants. *Partial Belief and Probabilistic Reasoning in the Analysis of Secure Protocols*, Proc. IEEE Computer Security Foundation Workshop V, New Hampshire, July 1992.
5. E. Campbell, and R. Safavi-Naini. *A Probabilistic Approach to the Analysis of Secure Protocols*, IFIP, May 1992.
6. E. Campbell, and R. Safavi-Naini. *On Automating the BAN Logic of Authentication*, Proc. of the Fifteenth Australian Computer Science Conference, 1992.

7. E. Campbell, R. Safavi-Naini, and P. Pleasants. *Probabilistic Logic and its Application to the Analysis of Protocol Security*, UNE Technical Report TR. 92-47, University of New England, Armidale.
8. R. Kailar, and V. D. Gligor. *On Belief Evolution in Authentication Protocols*, Proc. IEEE Computer Security Foundations Workshop IV, pages 103–116, 1991.
9. L. Gong, R. M. Needham, and R. Yahalom. *Reasoning about Belief in Cryptographic Protocols*, Proc. IEEE Symposium on Security and Privacy, pages 234–248, 1990.
10. CCITT. *Draft Recommendation X.509*, The Directory-Authentication Framework, Version 7, Gloucester.
11. D. Otway, and O. Rees. *Efficient and timely mutual authentication*, ACM Operating Systems Review, 21(1):8–10, January 1987.
12. R. M. Needham, and M. D. Schroeder. *Authentication Revisited*, ACM Operating Systems Review, 21(1):7, January 1987.
13. S. P. Miller, C. Neuman, J. I. Schiller, and J. H. Saltzer. *Kerberos Authentication and Authorization System*, Project Athena Technical Plan, Sect. E.2.1. MIT, Cambridge, Mass., July 1987.
14. A. D. Birrell. *Secure communication using remote procedure calls*, ACM Trans. on Computing Systems, 3(1):1–14, February 1985.
15. D. E. Denning, and G. M. Sacco. *Timestamps in Key Distribution Protocols*, Communications of the ACM, 24(8):533–536, August 1981.
16. R. M. Needham, and M. D. Schroeder, *Using Encryption for Authentication in Large Networks of Computers*, Communications of the ACM, 21(12):993–999, December 1978.
17. J. A. Thom, and J. Zobel (Eds.), *The NU-Prolog Reference Manual*, Technical Report 86/10, Dept. of Computer Science, University of Melbourne, 1987.

18. K. F. Sagonas, T. Swift, and D. S. Warren, *The XSB Programmer's Manual Version 1.3*, Dept. of Computer Science, SUNY, Stony Brook, New York, September 1993.
19. L. Sterling, and E. Shapiro. *The Art of Prolog*. MIT Press, 1986.

Appendix B

Examples of Machine-aided BAN Analysis

B.1 The Otway-Rees Protocol

Message 1 $A \rightarrow B$: $M, A, B, \{N_a, M, A, B\}_{K_{as}}$

Message 2 $B \rightarrow S$: $M, A, B, \{N_a, M, A, B\}_{K_{as}}, \{N_b, M, A, B\}_{K_{bs}}$

Message 3 $S \rightarrow B$: $M, \{N_a, K_{ab}\}_{K_{as}}, \{N_b, K_{ab}\}_{K_{bs}}$

Message 4 $B \rightarrow A$: $M, \{N_a, K_{ab}\}_{K_{as}}$

B.1.1 Program Representation

% The Otway-Rees Protocol

% Idealized Protocol

```
fact(1, sends(a, b, encrypt(kas, [na, nc])), reason([], 'Step')).
fact(2, sends(b, s, [encrypt(kas, [na, nc]), encrypt(kbs, [nb, nc])]),
reason([], 'Step')).
fact(3, sends(s, b, [encrypt(kas, [na, goodkey(kab, a, b), oncesaid(b, nc)]),
encrypt(kbs, [nb, goodkey(kab, a, b), oncesaid(a, nc)])]), reason([], 'Step')).
fact(4, sends(b, a, encrypt(kas, [na, goodkey(kab, a, b), oncesaid(b, nc)])),
reason([], 'Step')).
```

% Assumptions

```
fact(5, believes(a, goodkey(kas, a, s)), reason([], 'Assumption')).
fact(6, believes(a, jurisdiction(s, goodkey(_, a, b))), reason([], 'Assumption')).
fact(7, believes(a, jurisdiction(s, oncesaid(b, _))), reason([], 'Assumption')).
fact(8, believes(a, fresh(na)), reason([], 'Assumption')).
fact(9, believes(a, fresh(nc)), reason([], 'Assumption')).
fact(10, believes(b, goodkey(kbs, b, s)), reason([], 'Assumption')).
```

```

fact(11, believes(b, jurisdiction(s, goodkey(_, a, b))), reason([], 'Assumption')).
fact(12, believes(b, jurisdiction(s, oncesaid(a, _))), reason([], 'Assumption')).
fact(13, believes(b, fresh(nb)), reason([], 'Assumption')).
fact(14, believes(s, goodkey(kas, a, s)), reason([], 'Assumption')).
fact(15, believes(s, goodkey(kbs, b, s)), reason([], 'Assumption')).
fact(16, believes(s, goodkey(kab, a, b)), reason([], 'Assumption')).

```

B.1.2 Proofs

```
| ?- analyze(or).
```

Analyzed in 4 cycles

yes

```

| ?- explain_proof(believes(a, goodkey(kab, a, b))).
1. sends(b,a,encrypt(kas,[na,goodkey(kab,a,b),oncesaid(b,nc)])) {Step}
2. believes(a,goodkey(kas,a,s)) {Assumption}
3. sees(a,encrypt(kas,[na,goodkey(kab,a,b),oncesaid(b,nc)])) {1, Seeing}
4. believes(a,goodkey(kas,s,a)) {2, Bidirectionality}
5. believes(a,fresh(na)) {Assumption}
6. believes(a,oncesaid(s,[na,goodkey(kab,a,b),oncesaid(b,nc)])) {4, 3, Message meaning}
7. believes(a,fresh([na,goodkey(kab,a,b),oncesaid(b,nc)])) {5, Freshness}
8. believes(a,believes(s,[na,goodkey(kab,a,b),oncesaid(b,nc)])) {7, 6, Nonce verification}
9. believes(a,believes(s,goodkey(kab,a,b))) {8, Belief}
10. believes(a,jurisdiction(s,goodkey(_1259444,a,b))) {Assumption}
11. believes(a,goodkey(kab,a,b)) {10, 9, Jurisdiction}

```

yes

```

| ?- explain_proof(believes(a, believes(b, nc))).
1. sends(b,a,encrypt(kas,[na,goodkey(kab,a,b),oncesaid(b,nc)])) {Step}
2. believes(a,goodkey(kas,a,s)) {Assumption}
3. sees(a,encrypt(kas,[na,goodkey(kab,a,b),oncesaid(b,nc)])) {1, Seeing}
4. believes(a,goodkey(kas,s,a)) {2, Bidirectionality}
5. believes(a,fresh(na)) {Assumption}
6. believes(a,oncesaid(s,[na,goodkey(kab,a,b),oncesaid(b,nc)])) {4, 3, Message meaning}
7. believes(a,fresh([na,goodkey(kab,a,b),oncesaid(b,nc)])) {5, Freshness}
8. believes(a,believes(s,[na,goodkey(kab,a,b),oncesaid(b,nc)])) {7, 6, Nonce verification}
9. believes(a,believes(s,oncesaid(b,nc))) {8, Belief}
10. believes(a,jurisdiction(s,oncesaid(b,_1259604))) {Assumption}
11. believes(a,oncesaid(b,nc)) {10, 9, Jurisdiction}
12. believes(a,fresh(nc)) {Assumption}
13. believes(a,believes(b,nc)) {12, 11, Nonce verification}

```

yes

```

| ?- explain_proof(believes(b, goodkey(kab, a, b))).
1. sends(s,b,[encrypt(kas,[na,goodkey(kab,a,b),oncesaid(b,nc)]),encrypt(kbs,[nb,goodkey(kab,a,b),
oncesaid(a,nc)])]) {Step}
2. sees(b,[encrypt(kas,[na,goodkey(kab,a,b),oncesaid(b,nc)]),encrypt(kbs,[nb,goodkey(kab,a,b),

```

```

oncesaid(a,nc)])) {1, Seeing}
3. believes(b,goodkey(kbs,b,s)) {Assumption}
4. sees(b,encrypt(kbs,[nb,goodkey(kab,a,b),oncesaid(a,nc)])) {2, Sight}
5. believes(b,goodkey(kbs,s,b)) {3, Bidirectionality}
6. believes(b,fresh(nb)) {Assumption}
7. believes(b,oncesaid(s,[nb,goodkey(kab,a,b),oncesaid(a,nc)])) {5, 4, Message meaning}
8. believes(b,fresh([nb,goodkey(kab,a,b),oncesaid(a,nc)])) {6, Freshness}
9. believes(b,believes(s,[nb,goodkey(kab,a,b),oncesaid(a,nc)])) {8, 7, Nonce verification}
10. believes(b,believes(s,goodkey(kab,a,b))) {9, Belief}
11. believes(b,jurisdiction(s,goodkey(_1260024,a,b))) {Assumption}
12. believes(b,goodkey(kab,a,b)) {11, 10, Jurisdiction}

```

yes

```

| ?- explain_proof(believes(b, oncesaid(a, nc))).
1. sends(s,b,[encrypt(kas,[na,goodkey(kab,a,b),oncesaid(b,nc)]),encrypt(kbs,[nb,goodkey(kab,a,b),
oncesaid(a,nc)])) {Step}
2. sees(b,[encrypt(kas,[na,goodkey(kab,a,b),oncesaid(b,nc)]),encrypt(kbs,[nb,goodkey(kab,a,b),
oncesaid(a,nc)])) {1, Seeing}
3. believes(b,goodkey(kbs,b,s)) {Assumption}
4. sees(b,encrypt(kbs,[nb,goodkey(kab,a,b),oncesaid(a,nc)])) {2, Sight}
5. believes(b,goodkey(kbs,s,b)) {3, Bidirectionality}
6. believes(b,fresh(nb)) {Assumption}
7. believes(b,oncesaid(s,[nb,goodkey(kab,a,b),oncesaid(a,nc)])) {5, 4, Message meaning}
8. believes(b,fresh([nb,goodkey(kab,a,b),oncesaid(a,nc)])) {6, Freshness}
9. believes(b,believes(s,[nb,goodkey(kab,a,b),oncesaid(a,nc)])) {8, 7, Nonce verification}
10. believes(b,believes(s,oncesaid(a,nc))) {9, Belief}
11. believes(b,jurisdiction(s,oncesaid(a,_1259964))) {Assumption}
12. believes(b,oncesaid(a,nc)) {11, 10, Jurisdiction}

```

yes

```

| ?- [user].
[Compiling user]
goal(believes(a, goodkey(kab, a, b))).
goal(believes(a, believes(b, nc))).
goal(believes(b, goodkey(kab, a, b))).
goal(believes(b, oncesaid(a, nc))).
[user compiled, cpu time used: 0.86101 seconds]
[user loaded]

```

yes

```

| ?- chk_red(a, A, As).

```

no

```

| ?- chk_red(b, A, As).

```

no

B.2 The Needham-Schroeder Shared-Key Protocol

Message 1 $A \rightarrow S : A, B, N_a$

Message 2 $S \rightarrow A : \{N_a, B, K_{ab}, \{K_{ab}, A\}_{K_{bs}}\}_{K_{as}}$

Message 3 $A \rightarrow B : \{K_{ab}, A\}_{K_{bs}}$

Message 4 $B \rightarrow A : \{N_b\}_{K_{ab}}$

Message 5 $A \rightarrow B : \{N_b - 1\}_{K_{ab}}$

B.2.1 Program Representation

% The Needham-Schroeder Shared-Key Protocol

% Idealized protocol

```
fact(1, sends(s, a, encrypt(kas, [na, goodkey(kab, a, b), fresh(goodkey(kab, a, b)),
encrypt(kbs, goodkey(kab, a, b)])), reason([], 'Step'))).
fact(2, sends(a, b, encrypt(kbs, goodkey(kab, a, b))), reason([], 'Step')).
fact(3, sends(b, a, encrypt(kab, [nb, goodkey(kab, a, b)])), reason([], 'Step')).
fact(4, sends(a, b, encrypt(kab, [nb, goodkey(kab, a, b)])), reason([], 'Step')).
```

% Assumptions

```
fact(5, believes(a, goodkey(kas, a, s)), reason([], 'Assumption')).
fact(6, believes(a, jurisdiction(s, goodkey(_, a, b))), reason([], 'Assumption')).
fact(7, believes(a, jurisdiction(s, fresh(goodkey(_, a, b)))), reason([], 'Assumption')).
fact(8, believes(a, fresh(na)), reason([], 'Assumption')).
fact(9, believes(b, goodkey(kbs, b, s)), reason([], 'Assumption')).
fact(10, believes(b, jurisdiction(s, goodkey(_, a, b))), reason([], 'Assumption')).
fact(11, believes(b, fresh(nb)), reason([], 'Assumption')).
fact(12, believes(b, fresh(goodkey(kab, a, b))), reason([], 'Assumption')).
fact(13, believes(s, goodkey(kas, a, s)), reason([], 'Assumption')).
fact(14, believes(s, goodkey(kbs, b, s)), reason([], 'Assumption')).
fact(15, believes(s, goodkey(kab, a, b)), reason([], 'Assumption')).
fact(16, believes(s, fresh(goodkey(kab, a, b))), reason([], 'Assumption')).
```

B.2.2 Proofs

| ?- analyze(nssk).

Analyzed in 6 cycles

yes

```
| ?- explain_proof(believes(a, believes(b, goodkey(kab, a, b)))).
1. sends(b,a,encrypt(kab,[nb,goodkey(kab,a,b)])) {Step}
2. sends(s,a,encrypt(kas,[na,goodkey(kab,a,b),fresh(goodkey(kab,a,b)),
encrypt(kbs,goodkey(kab,a,b)])) {Step}
3. believes(a,goodkey(kas,a,s)) {Assumption}
```


4. sees(a,encrypt(kas,[na,goodkey(kab,a,b),fresh(goodkey(kab,a,b)),
encrypt(kbs,goodkey(kab,a,b))])) {2, Seeing}
5. believes(a,goodkey(kas,s,a)) {3, Bidirectionality}
6. believes(a,fresh(na)) {Assumption}
7. believes(a,oncesaid(s,[na,goodkey(kab,a,b),fresh(goodkey(kab,a,b)),encrypt(kbs,goodkey(kab,a,b))]))
{5, 4, Message meaning}
8. believes(a,fresh([na,goodkey(kab,a,b),fresh(goodkey(kab,a,b)),encrypt(kbs,goodkey(kab,a,b))]))
{6, Freshness}
9. believes(a,believes(s,[na,goodkey(kab,a,b),fresh(goodkey(kab,a,b)),encrypt(kbs,goodkey(kab,a,b))]))
{8, 7, Nonce verification}
10. believes(a,believes(s,goodkey(kab,a,b))) {9, Belief}
11. believes(a,jurisdiction(s,goodkey(_1262876,a,b))) {Assumption}
12. believes(a,goodkey(kab,a,b)) {11, 10, Jurisdiction}
13. sees(a,encrypt(kab,[nb,goodkey(kab,a,b)])) {1, Seeing}
14. believes(a,goodkey(kab,b,a)) {12, Bidirectionality}
15. believes(a,believes(s,fresh(goodkey(kab,a,b)))) {9, Belief}
16. believes(a,jurisdiction(s,fresh(goodkey(_1263184,a,b)))) {Assumption}
17. believes(a,fresh(goodkey(kab,a,b))) {16, 15, Jurisdiction}
18. believes(a,oncesaid(b,[nb,goodkey(kab,a,b)])) {14, 13, Message meaning}
19. believes(a,fresh([nb,goodkey(kab,a,b)])) {17, Freshness}
20. believes(a,believes(b,[nb,goodkey(kab,a,b)])) {19, 18, Nonce verification}
21. believes(a,believes(b,goodkey(kab,a,b))) {20, Belief}

1. sends(b,a,encrypt(kab,[nb,goodkey(kab,a,b)])) {Step}
2. sends(s,a,encrypt(kas,[na,goodkey(kab,a,b),fresh(goodkey(kab,a,b)),encrypt(kbs,goodkey(kab,a,b))]))
{Step}
3. believes(a,goodkey(kas,a,s)) {Assumption}
4. sees(a,encrypt(kas,[na,goodkey(kab,a,b),fresh(goodkey(kab,a,b)),encrypt(kbs,goodkey(kab,a,b))]))
{2, Seeing}
5. believes(a,goodkey(kas,s,a)) {3, Bidirectionality}
6. believes(a,fresh(na)) {Assumption}
7. believes(a,oncesaid(s,[na,goodkey(kab,a,b),fresh(goodkey(kab,a,b)),encrypt(kbs,goodkey(kab,a,b))]))
{5, 4, Message meaning}
8. believes(a,fresh([na,goodkey(kab,a,b),fresh(goodkey(kab,a,b)),encrypt(kbs,goodkey(kab,a,b))]))
{6, Freshness}
9. believes(a,believes(s,[na,goodkey(kab,a,b),fresh(goodkey(kab,a,b)),encrypt(kbs,goodkey(kab,a,b))]))
{8, 7, Nonce verification}
10. believes(a,believes(s,goodkey(kab,a,b))) {9, Belief}
11. believes(a,jurisdiction(s,goodkey(_1262584,a,b))) {Assumption}
12. believes(a,goodkey(kab,a,b)) {11, 10, Jurisdiction}
13. sees(a,encrypt(kab,[nb,goodkey(kab,a,b)])) {1, Seeing}
14. believes(a,goodkey(kab,b,a)) {12, Bidirectionality}
15. believes(a,oncesaid(b,[nb,goodkey(kab,a,b)])) {14, 13, Message meaning}
16. believes(a,believes(s,fresh(goodkey(kab,a,b)))) {9, Belief}
17. believes(a,jurisdiction(s,fresh(goodkey(_1262976,a,b)))) {Assumption}
18. believes(a,oncesaid(b,goodkey(kab,a,b))) {15, Utterance}
19. believes(a,fresh(goodkey(kab,a,b))) {17, 16, Jurisdiction}
20. believes(a,believes(b,goodkey(kab,a,b))) {19, 18, Nonce verification}

1. sends(b,a,encrypt(kab,[nb,goodkey(kab,a,b)])) {Step}

2. sends(s,a,encrypt(kas,[na,goodkey(kab,a,b),fresh(goodkey(kab,a,b)),encrypt(kbs,goodkey(kab,a,b))]))
 {Step}
 3. believes(a,goodkey(kas,a,s)) {Assumption}
 4. sees(a,encrypt(kas,[na,goodkey(kab,a,b),fresh(goodkey(kab,a,b)),encrypt(kbs,goodkey(kab,a,b))]))
 {2, Seeing}
 5. believes(a,goodkey(kas,s,a)) {3, Bidirectionality}
 6. believes(a,oncesaid(s,[na,goodkey(kab,a,b),fresh(goodkey(kab,a,b)),encrypt(kbs,goodkey(kab,a,b))]))
 {5, 4, Message meaning}
 7. believes(a,fresh(na)) {Assumption}
 8. believes(a,fresh([na,goodkey(kab,a,b),fresh(goodkey(kab,a,b)),encrypt(kbs,goodkey(kab,a,b))]))
 {7, Freshness}
 9. believes(a,believes(s,[na,goodkey(kab,a,b),fresh(goodkey(kab,a,b)),encrypt(kbs,goodkey(kab,a,b))]))
 {8, 6, Nonce verification}
 10. believes(a,believes(s,fresh(goodkey(kab,a,b)))) {9, Belief}
 11. believes(a,jurisdiction(s,fresh(goodkey(_1264392,a,b)))) {Assumption}
 12. believes(a,oncesaid(s,goodkey(kab,a,b))) {6, Utterance}
 13. believes(a,fresh(goodkey(kab,a,b))) {11, 10, Jurisdiction}
 14. believes(a,believes(s,goodkey(kab,a,b))) {13, 12, Nonce verification}
 15. believes(a,jurisdiction(s,goodkey(_1264636,a,b))) {Assumption}
 16. believes(a,goodkey(kab,a,b)) {15, 14, Jurisdiction}
 17. sees(a,encrypt(kab,[nb,goodkey(kab,a,b)])) {1, Seeing}
 18. believes(a,goodkey(kab,b,a)) {16, Bidirectionality}
 19. believes(a,oncesaid(b,[nb,goodkey(kab,a,b)])) {18, 17, Message meaning}
 20. believes(a,fresh([nb,goodkey(kab,a,b)])) {13, Freshness}
 21. believes(a,believes(b,[nb,goodkey(kab,a,b)])) {20, 19, Nonce verification}
 22. believes(a,believes(b,goodkey(kab,a,b))) {21, Belief}

1. sends(b,a,encrypt(kab,[nb,goodkey(kab,a,b)])) {Step}
 2. sends(s,a,encrypt(kas,[na,goodkey(kab,a,b),fresh(goodkey(kab,a,b)),encrypt(kbs,goodkey(kab,a,b))]))
 {Step}
 3. believes(a,goodkey(kas,a,s)) {Assumption}
 4. sees(a,encrypt(kas,[na,goodkey(kab,a,b),fresh(goodkey(kab,a,b)),encrypt(kbs,goodkey(kab,a,b))]))
 {2, Seeing}
 5. believes(a,goodkey(kas,s,a)) {3, Bidirectionality}
 6. believes(a,oncesaid(s,[na,goodkey(kab,a,b),fresh(goodkey(kab,a,b)),encrypt(kbs,goodkey(kab,a,b))]))
 {5, 4, Message meaning}
 7. believes(a,fresh(na)) {Assumption}
 8. believes(a,fresh([na,goodkey(kab,a,b),fresh(goodkey(kab,a,b)),encrypt(kbs,goodkey(kab,a,b))]))
 {7, Freshness}
 9. believes(a,believes(s,[na,goodkey(kab,a,b),fresh(goodkey(kab,a,b)),encrypt(kbs,goodkey(kab,a,b))]))
 {8, 6, Nonce verification}
 10. believes(a,believes(s,fresh(goodkey(kab,a,b)))) {9, Belief}
 11. believes(a,jurisdiction(s,fresh(goodkey(_1264100,a,b)))) {Assumption}
 12. believes(a,oncesaid(s,goodkey(kab,a,b))) {6, Utterance}
 13. believes(a,fresh(goodkey(kab,a,b))) {11, 10, Jurisdiction}
 14. believes(a,believes(s,goodkey(kab,a,b))) {13, 12, Nonce verification}
 15. believes(a,jurisdiction(s,goodkey(_1264344,a,b))) {Assumption}
 16. believes(a,goodkey(kab,a,b)) {15, 14, Jurisdiction}
 17. sees(a,encrypt(kab,[nb,goodkey(kab,a,b)])) {1, Seeing}
 18. believes(a,goodkey(kab,b,a)) {16, Bidirectionality}

19. believes(a,oncesaid(b,[nb,goodkey(kab,a,b)])) {18, 17, Message meaning}
20. believes(a,oncesaid(b,goodkey(kab,a,b))) {19, Utterance}
21. believes(a,believes(b,goodkey(kab,a,b))) {13, 20, Nonce verification}

yes

| ?- explain_proof(believes(b, believes(a, goodkey(kab, a, b)))).

1. sends(a,b,encrypt(kab,[nb,goodkey(kab,a,b)])) {Step}
2. sends(a,b,encrypt(kbs,goodkey(kab,a,b))) {Step}
3. believes(b,goodkey(kbs,b,s)) {Assumption}
4. sees(b,encrypt(kbs,goodkey(kab,a,b))) {2, Seeing}
5. believes(b,goodkey(kbs,s,b)) {3, Bidirectionality}
6. believes(b,oncesaid(s,goodkey(kab,a,b))) {5, 4, Message meaning}
7. believes(b,fresh(goodkey(kab,a,b))) {Assumption}
8. believes(b,believes(s,goodkey(kab,a,b))) {7, 6, Nonce verification}
9. believes(b,jurisdiction(s,goodkey(_1259896,a,b))) {Assumption}
10. sees(b,encrypt(kab,[nb,goodkey(kab,a,b)])) {1, Seeing}
11. believes(b,goodkey(kab,a,b)) {9, 8, Jurisdiction}
12. believes(b,fresh(nb)) {Assumption}
13. believes(b,oncesaid(a,[nb,goodkey(kab,a,b)])) {11, 10, Message meaning}
14. believes(b,fresh([nb,goodkey(kab,a,b)])) {12, Freshness}
15. believes(b,believes(a,[nb,goodkey(kab,a,b)])) {14, 13, Nonce verification}
16. believes(b,believes(a,goodkey(kab,a,b))) {15, Belief}

1. sends(a,b,encrypt(kab,[nb,goodkey(kab,a,b)])) {Step}
2. sends(a,b,encrypt(kbs,goodkey(kab,a,b))) {Step}
3. believes(b,goodkey(kbs,b,s)) {Assumption}
4. sees(b,encrypt(kbs,goodkey(kab,a,b))) {2, Seeing}
5. believes(b,goodkey(kbs,s,b)) {3, Bidirectionality}
6. believes(b,oncesaid(s,goodkey(kab,a,b))) {5, 4, Message meaning}
7. believes(b,fresh(goodkey(kab,a,b))) {Assumption}
8. believes(b,believes(s,goodkey(kab,a,b))) {7, 6, Nonce verification}
9. believes(b,jurisdiction(s,goodkey(_1259896,a,b))) {Assumption}
10. sees(b,encrypt(kab,[nb,goodkey(kab,a,b)])) {1, Seeing}
11. believes(b,goodkey(kab,a,b)) {9, 8, Jurisdiction}
12. believes(b,oncesaid(a,[nb,goodkey(kab,a,b)])) {11, 10, Message meaning}
13. believes(b,fresh([nb,goodkey(kab,a,b)])) {7, Freshness}
14. believes(b,believes(a,[nb,goodkey(kab,a,b)])) {13, 12, Nonce verification}
15. believes(b,believes(a,goodkey(kab,a,b))) {14, Belief}

1. sends(a,b,encrypt(kab,[nb,goodkey(kab,a,b)])) {Step}
2. sends(a,b,encrypt(kbs,goodkey(kab,a,b))) {Step}
3. believes(b,goodkey(kbs,b,s)) {Assumption}
4. sees(b,encrypt(kbs,goodkey(kab,a,b))) {2, Seeing}
5. believes(b,goodkey(kbs,s,b)) {3, Bidirectionality}
6. believes(b,oncesaid(s,goodkey(kab,a,b))) {5, 4, Message meaning}
7. believes(b,fresh(goodkey(kab,a,b))) {Assumption}
8. believes(b,believes(s,goodkey(kab,a,b))) {7, 6, Nonce verification}
9. believes(b,jurisdiction(s,goodkey(_1259764,a,b))) {Assumption}
10. sees(b,encrypt(kab,[nb,goodkey(kab,a,b)])) {1, Seeing}

```

11. believes(b,goodkey(kab,a,b)) {9, 8, Jurisdiction}
12. believes(b,oncesaid(a,[nb,goodkey(kab,a,b)])) {11, 10, Message meaning}
13. believes(b,oncesaid(a,goodkey(kab,a,b))) {12, Utterance}
14. believes(b,believes(a,goodkey(kab,a,b))) {7, 13, Nonce verification}

```

yes

| ?- [user].

[Compiling user]

goal(believes(a, goodkey(kab, a, b))).

goal(believes(a, believes(b, goodkey(kab, a, b)))).

goal(believes(b, goodkey(kab, a, b))).

goal(believes(b, believes(a, goodkey(kab, a, b)))).

[user compiled, cpu time used: 0.29102 seconds]

[user loaded]

yes

| ?- chk_red(a, A, As).

no

| ?- chk_red(b, A, As).

A = believes(b,fresh(nb))

As = [believes(b,goodkey(kbs,b,s)),believes(b,jurisdiction(s,goodkey(_1257384,a,b))),
believes(b,fresh(goodkey(kab,a,b)))];

no

B.3 The Kerberos Protocol

Message 1 $A \rightarrow S : A, B$

Message 2 $S \rightarrow A : \{T_s, L, K_{ab}, B, \{T_s, L, K_{ab}, A\}_{K_{bs}}\}_{K_{as}}$

Message 3 $A \rightarrow B : \{T_s, L, K_{ab}, A\}_{K_{bs}}, \{A, T_a\}_{K_{ab}}$

Message 4 $B \rightarrow A : \{T_a + 1\}_{K_{ab}}$

B.3.1 Program Representation

% The Kerberos Protocol

% Idealized Protocol

fact(1, sends(s, a, encrypt(kas, [ts, goodkey(kab, a, b), encrypt(kbs, [ts, goodkey(kab, a, b)])))),
reason([], 'Step')).

fact(2, sends(a, b, [encrypt(kbs, [ts, goodkey(kab, a, b)]), encrypt(kab, [ta, goodkey(kab, a, b)]))),
reason([], 'Step')).

fact(3, sends(b, a, encrypt(kab, [ta, goodkey(kab, a, b)])), reason([], 'Step')).

```

% Assumptions
fact(4, believes(a, goodkey(kas, a, s)), reason([], 'Assumption')).
fact(5, believes(a, jurisdiction(s, goodkey(_, a, b))), reason([], 'Assumption')).
fact(6, believes(a, fresh(ts)), reason([], 'Assumption')).
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% This assumption is missing in the analysis given in [1]
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
fact(7, believes(a, fresh(ta)), reason([], 'Assumption')).
fact(8, believes(b, goodkey(kbs, b, s)), reason([], 'Assumption')).
fact(9, believes(b, jurisdiction(s, goodkey(_, a, b))), reason([], 'Assumption')).
fact(10, believes(b, fresh(ts)), reason([], 'Assumption')).
fact(11, believes(b, fresh(ta)), reason([], 'Assumption')).
fact(12, believes(s, goodkey(kas, a, s)), reason([], 'Assumption')).
fact(13, believes(s, goodkey(kbs, b, s)), reason([], 'Assumption')).
fact(14, believes(s, goodkey(kab, a, b)), reason([], 'Assumption')).

```

B.3.2 Proofs

```

| ?- analyze(kerberos).
Analyzed in 6 cycles

```

```

yes
| ?- explain_proof(believes(a, goodkey(kab, a, b))).
1. sends(s,a,encrypt(kas,[ts,goodkey(kab,a,b),encrypt(kbs,[ts,goodkey(kab,a,b)])])) {Step}
2. believes(a,goodkey(kas,a,s)) {Assumption}
3. sees(a,encrypt(kas,[ts,goodkey(kab,a,b),encrypt(kbs,[ts,goodkey(kab,a,b)])])) {1, Seeing}
4. believes(a,goodkey(kas,s,a)) {2, Bidirectionality}
5. believes(a,fresh(ts)) {Assumption}
6. believes(a,oncesaid(s,[ts,goodkey(kab,a,b),encrypt(kbs,[ts,goodkey(kab,a,b)])]))
{4, 3, Message meaning}
7. believes(a,fresh([ts,goodkey(kab,a,b),encrypt(kbs,[ts,goodkey(kab,a,b)])])) {5, Freshness}
8. believes(a,believes(s,[ts,goodkey(kab,a,b),encrypt(kbs,[ts,goodkey(kab,a,b)])])) {
7, 6, Nonce verification}
9. believes(a,believes(s,goodkey(kab,a,b))) {8, Belief}
10. believes(a,jurisdiction(s,goodkey(_1259764,a,b))) {Assumption}
11. believes(a,goodkey(kab,a,b)) {10, 9, Jurisdiction}

```

```

yes

```

The need for the assumption $A \models \sharp(T_a)$ is highlighted by the following proof of the second-order goal of A :

```

| ?- explain_proof(believes(a, believes(b, goodkey(kab, a, b)))).
1. sends(b,a,encrypt(kab,[ta,goodkey(kab,a,b)])) {Step}
2. sends(s,a,encrypt(kas,[ts,goodkey(kab,a,b),encrypt(kbs,[ts,goodkey(kab,a,b)])])) {Step}
3. believes(a,goodkey(kas,a,s)) {Assumption}
4. sees(a,encrypt(kas,[ts,goodkey(kab,a,b),encrypt(kbs,[ts,goodkey(kab,a,b)])])) {2, Seeing}

```

```

5. believes(a,goodkey(kas,s,a)) {3, Bidirectionality}
6. believes(a,fresh(ts)) {Assumption}
7. believes(a,oncesaid(s,[ts,goodkey(kab,a,b),encrypt(kbs,[ts,goodkey(kab,a,b)])]))
{5, 4, Message meaning}
8. believes(a,fresh([ts,goodkey(kab,a,b),encrypt(kbs,[ts,goodkey(kab,a,b)])])) {6, Freshness}
9. believes(a,believes(s,[ts,goodkey(kab,a,b),encrypt(kbs,[ts,goodkey(kab,a,b)])]))
{8, 7, Nonce verification}
10. believes(a,believes(s,goodkey(kab,a,b))) {9, Belief}
11. believes(a,jurisdiction(s,goodkey(_1261140,a,b))) {Assumption}
12. believes(a,goodkey(kab,a,b)) {11, 10, Jurisdiction}
13. sees(a,encrypt(kab,[ta,goodkey(kab,a,b)])) {1, Seeing}
14. believes(a,goodkey(kab,b,a)) {12, Bidirectionality}
15. believes(a,fresh(ta)) {Assumption}
16. believes(a,oncesaid(b,[ta,goodkey(kab,a,b)])) {14, 13, Message meaning}
17. believes(a,fresh([ta,goodkey(kab,a,b)])) {15, Freshness}
18. believes(a,believes(b,[ta,goodkey(kab,a,b)])) {17, 16, Nonce verification}
19. believes(a,believes(b,goodkey(kab,a,b))) {18, Belief}

```

yes

```

| ?- explain_proof(believes(b, goodkey(kab, a, b))).
1. sends(a,b,[encrypt(kbs,[ts,goodkey(kab,a,b)],encrypt(kab,[ta,goodkey(kab,a,b)]))] {Step}
2. sees(b,[encrypt(kbs,[ts,goodkey(kab,a,b)],encrypt(kab,[ta,goodkey(kab,a,b)]))] {1, Seeing}
3. believes(b,goodkey(kbs,b,s)) {Assumption}
4. sees(b,encrypt(kbs,[ts,goodkey(kab,a,b)])) {2, Sight}
5. believes(b,goodkey(kbs,s,b)) {3, Bidirectionality}
6. believes(b,fresh(ts)) {Assumption}
7. believes(b,oncesaid(s,[ts,goodkey(kab,a,b)])) {5, 4, Message meaning}
8. believes(b,fresh([ts,goodkey(kab,a,b)])) {6, Freshness}
9. believes(b,believes(s,[ts,goodkey(kab,a,b)])) {8, 7, Nonce verification}
10. believes(b,believes(s,goodkey(kab,a,b))) {9, Belief}
11. believes(b,jurisdiction(s,goodkey(_1259704,a,b))) {Assumption}
12. believes(b,goodkey(kab,a,b)) {11, 10, Jurisdiction}

```

yes

```

| ?- explain_proof(believes(b, believes(a, goodkey(kab, a, b)))).
1. sends(a,b,[encrypt(kbs,[ts,goodkey(kab,a,b)],encrypt(kab,[ta,goodkey(kab,a,b)]))] {Step}
2. sees(b,[encrypt(kbs,[ts,goodkey(kab,a,b)],encrypt(kab,[ta,goodkey(kab,a,b)]))] {1, Seeing}
3. believes(b,goodkey(kbs,b,s)) {Assumption}
4. sees(b,encrypt(kbs,[ts,goodkey(kab,a,b)])) {2, Sight}
5. believes(b,goodkey(kbs,s,b)) {3, Bidirectionality}
6. believes(b,fresh(ts)) {Assumption}
7. believes(b,oncesaid(s,[ts,goodkey(kab,a,b)])) {5, 4, Message meaning}
8. believes(b,fresh([ts,goodkey(kab,a,b)])) {6, Freshness}
9. believes(b,believes(s,[ts,goodkey(kab,a,b)])) {8, 7, Nonce verification}
10. believes(b,believes(s,goodkey(kab,a,b))) {9, Belief}
11. believes(b,jurisdiction(s,goodkey(_1261076,a,b))) {Assumption}
12. sees(b,encrypt(kab,[ta,goodkey(kab,a,b)])) {2, Sight}

```

```

13. believes(b,goodkey(kab,a,b)) {11, 10, Jurisdiction}
14. believes(b,fresh(ta)) {Assumption}
15. believes(b,oncesaid(a,[ta,goodkey(kab,a,b)])) {13, 12, Message meaning}
16. believes(b,fresh([ta,goodkey(kab,a,b)])) {14, Freshness}
17. believes(b,believes(a,[ta,goodkey(kab,a,b)])) {16, 15, Nonce verification}
18. believes(b,believes(a,goodkey(kab,a,b))) {17, Belief}

```

```

yes
| ?- [user].
[Compiling user]
goal(believes(a, goodkey(kab, a, b))).
goal(believes(a, believes(b, goodkey(kab, a, b)))).
goal(believes(b, goodkey(kab, a, b))).
goal(believes(b, believes(a, goodkey(kab, a, b)))).
[user compiled, cpu time used: 0.311 seconds]
[user loaded]

```

```

yes
| ?- chk_red(a, A, As).

```

```

no
| ?- chk_red(b, A, As).

```

```

no

```

B.4 The Yahalom Protocol

Message 1 $A \rightarrow B : A, N_a$
 Message 2 $B \rightarrow S : B, \{A, N_a, N_b\}_{K_{bs}}$
 Message 3 $S \rightarrow A : \{B, K_{ab}, N_a, N_b\}_{K_{as}}, \{A, K_{ab}\}_{K_{bs}}$
 Message 4 $A \rightarrow B : \{A, K_{ab}\}_{K_{bs}}, \{N_b\}_{K_{ab}}$

B.4.1 Program Representation

```

fact(1, sends(b, s, encrypt(kbs, [na, nb])), reason([], 'Step')).
fact(2, sends(s, a, [encrypt(kas, [goodkey(kab, a, b), fresh(goodkey(kab, a, b))
, na, nb, oncesaid(b, na)]), encrypt(kbs, goodkey(kab, a, b))]), reason([], 'Step')).
fact(3, sends(a, b, [encrypt(kbs, goodkey(kab, a, b)), encrypt(kab, combined(nb,
[nb, goodkey(kab, a, b), believes(s, fresh(goodkey(kab, a, b))]))])), reason([],
'Step')).
% Assumptions
fact(4, believes(a, goodkey(kas, a, s)), reason([], 'Assumption')).
fact(5, believes(a, jurisdiction(s, goodkey(_, a, b))), reason([], 'Assumption')).

```

```

fact(6, believes(a, fresh(na)), reason([], 'Assumption')).
fact(7, believes(a, jurisdiction(s, oncesaid(b, _))), reason([], 'Assumption')).
fact(8, believes(b, goodkey(kbs, b, s)), reason([], 'Assumption')).
fact(9, believes(b, jurisdiction(s, goodkey(_, a, b))), reason([], 'Assumption')
).
fact(10, believes(b, fresh(nb)), reason([], 'Assumption')).
fact(11, believes(b, jurisdiction(s, fresh(goodkey(_, a, b)))), reason([], 'Assu
mption')).
fact(12, believes(b, jurisdiction(a, believes(s, fresh(goodkey(_, a, b)))),
reason([], 'Assumption')).
fact(13, believes(b, secret(nb, a, b)), reason([], 'Assumption')).
fact(14, believes(s, goodkey(kas, a, s)), reason([], 'Assumption')).
fact(15, believes(s, goodkey(kbs, b, s)), reason([], 'Assumption')).
fact(16, believes(s, goodkey(kab, a, b)), reason([], 'Assumption')).
fact(17, believes(s, fresh(goodkey(kab, a, b))), reason([], 'Assumption')).

```

B.4.2 Proofs

```
| ?- analyze(yahalom).
```

Analyzed in 7 cycles

yes

```

| ?- explain_proof(believes(a, goodkey(kab, a, b))).
1. sends(s,a,[encrypt(kas,[goodkey(kab,a,b),fresh(goodkey(kab,a,b)),na,nb,oncesaid(b,na)]),
encrypt(kbs,goodkey(kab,a,b))]) {Step}
2. sees(a,[encrypt(kas,[goodkey(kab,a,b),fresh(goodkey(kab,a,b)),na,nb,oncesaid(b,na)]),
encrypt(kbs,goodkey(kab,a,b))]) {1, Seeing}
3. believes(a,goodkey(kas,a,s)) {Assumption}
4. sees(a,encrypt(kas,[goodkey(kab,a,b),fresh(goodkey(kab,a,b)),na,nb,oncesaid(b,na)]))
{2, Sight}
5. believes(a,goodkey(kas,s,a)) {3, Bidirectionality}
6. believes(a,fresh(na)) {Assumption}
7. believes(a,oncesaid(s,[goodkey(kab,a,b),fresh(goodkey(kab,a,b)),na,nb,oncesaid(b,na)]))
{5, 4, Message meaning}
8. believes(a,fresh([goodkey(kab,a,b),fresh(goodkey(kab,a,b)),na,nb,oncesaid(b,na)]))
{6, Freshness}
9. believes(a,believes(s,[goodkey(kab,a,b),fresh(goodkey(kab,a,b)),na,nb,oncesaid(b,na)]))
{8, 7, Nonce verification}
10. believes(a,believes(s,goodkey(kab,a,b))) {9, Belief}
11. believes(a,jurisdiction(s,goodkey(_1260360,a,b))) {Assumption}
12. believes(a,goodkey(kab,a,b)) {11, 10, Jurisdiction}

```

yes

```

| ?- explain_proof(believes(a, believes(b, na))).
1. sends(s,a,[encrypt(kas,[goodkey(kab,a,b),fresh(goodkey(kab,a,b)),na,nb,oncesaid(b,na)]),
encrypt(kbs,goodkey(kab,a,b))]) {Step}
2. sees(a,[encrypt(kas,[goodkey(kab,a,b),fresh(goodkey(kab,a,b)),na,nb,oncesaid(b,na)]),
encrypt(kbs,goodkey(kab,a,b))]) {1, Seeing}

```



```

3. believes(a,goodkey(kas,a,s)) {Assumption}
4. sees(a,encrypt(kas,[goodkey(kab,a,b),fresh(goodkey(kab,a,b)),na,nb,oncesaid(b,na)]))
{2, Sight}
5. believes(a,goodkey(kas,s,a)) {3, Bidirectionality}
6. believes(a,fresh(na)) {Assumption}
7. believes(a,oncesaid(s,[goodkey(kab,a,b),fresh(goodkey(kab,a,b)),na,nb,oncesaid(b,na)]))
{5, 4, Message meaning}
8. believes(a,fresh([goodkey(kab,a,b),fresh(goodkey(kab,a,b)),na,nb,oncesaid(b,na)]))
{6, Freshness}
9. believes(a,believes(s,[goodkey(kab,a,b),fresh(goodkey(kab,a,b)),na,nb,oncesaid(b,na)]))
{8, 7, Nonce verification}
10. believes(a,believes(s,oncesaid(b,na))) {9, Belief}
11. believes(a,jurisdiction(s,oncesaid(b,_1260512))) {Assumption}
12. believes(a,oncesaid(b,na)) {11, 10, Jurisdiction}
13. believes(a,believes(b,na)) {6, 12, Nonce verification}

```

yes

```

| ?- explain_proof(believes(b, goodkey(kab, a, b))).
1. sends(a,b,[encrypt(kbs,goodkey(kab,a,b)),encrypt(kab,combined(nb,[nb,goodkey(kab,a,b),
believes(s,fresh(goodkey(kab,a,b))])))] {Step}
2. sees(b,[encrypt(kbs,goodkey(kab,a,b)),encrypt(kab,combined(nb,[nb,goodkey(kab,a,b),
believes(s,fresh(goodkey(kab,a,b))])))] {1, Seeing}
3. believes(b,goodkey(kbs,b,s)) {Assumption}
4. sees(b,encrypt(kbs,goodkey(kab,a,b))) {2, Sight}
5. believes(b,goodkey(kbs,s,b)) {3, Bidirectionality}
6. sees(b,encrypt(kab,combined(nb,[nb,goodkey(kab,a,b),believes(s,fresh(goodkey(kab,a,b))]))))
{2, Sight}
7. believes(b,oncesaid(s,goodkey(kab,a,b))) {5, 4, Message meaning}
8. sees(b,combined(nb,[nb,goodkey(kab,a,b),believes(s,fresh(goodkey(kab,a,b))]))
{7, 6, Message decryption}
9. believes(b,secret(nb,a,b)) {Assumption}
10. believes(b,fresh(nb)) {Assumption}
11. believes(b,oncesaid(a,[nb,goodkey(kab,a,b),believes(s,fresh(goodkey(kab,a,b))]))
{9, 8, Message meaning}
12. believes(b,fresh([nb,goodkey(kab,a,b),believes(s,fresh(goodkey(kab,a,b))])) {10, Freshness}
13. believes(b,believes(a,[nb,goodkey(kab,a,b),believes(s,fresh(goodkey(kab,a,b))]))
{12, 11, Nonce verification}
14. believes(b,believes(a,believes(s,fresh(goodkey(kab,a,b))))) {13, Belief}
15. believes(b,jurisdiction(a,believes(s,fresh(goodkey(_1263340,a,b))))) {Assumption}
16. believes(b,believes(s,fresh(goodkey(kab,a,b))) {15, 14, Jurisdiction}
17. believes(b,jurisdiction(s,fresh(goodkey(_1263476,a,b))) {Assumption}
18. believes(b,fresh(goodkey(kab,a,b))) {17, 16, Jurisdiction}
19. believes(b,believes(s,goodkey(kab,a,b))) {18, 7, Nonce verification}
20. believes(b,jurisdiction(s,goodkey(_1263660,a,b))) {Assumption}
21. believes(b,goodkey(kab,a,b)) {20, 19, Jurisdiction}

```

yes

```

| ?- explain_proof(believes(b, believes(a, goodkey(kab, a, b)))).

```

1. sends(a,b,[encrypt(kbs,goodkey(kab,a,b)),encrypt(kab,combined(nb,[nb,goodkey(kab,a,b),believes(s,fresh(goodkey(kab,a,b))])))] {Step}
2. sees(b,[encrypt(kbs,goodkey(kab,a,b)),encrypt(kab,combined(nb,[nb,goodkey(kab,a,b),believes(s,fresh(goodkey(kab,a,b))])))] {1, Seeing}
3. believes(b,goodkey(kbs,b,s)) {Assumption}
4. sees(b,encrypt(kbs,goodkey(kab,a,b))) {2, Sight}
5. believes(b,goodkey(kbs,s,b)) {3, Bidirectionality}
6. sees(b,encrypt(kab,combined(nb,[nb,goodkey(kab,a,b),believes(s,fresh(goodkey(kab,a,b))])))) {2, Sight}
7. believes(b,oncesaid(s,goodkey(kab,a,b))) {5, 4, Message meaning}
8. sees(b,combined(nb,[nb,goodkey(kab,a,b),believes(s,fresh(goodkey(kab,a,b))])) {7, 6, Message decryption}
9. believes(b,secret(nb,a,b)) {Assumption}
10. believes(b,fresh(nb)) {Assumption}
11. believes(b,oncesaid(a,[nb,goodkey(kab,a,b),believes(s,fresh(goodkey(kab,a,b))])) {9, 8, Message meaning}
12. believes(b,fresh([nb,goodkey(kab,a,b),believes(s,fresh(goodkey(kab,a,b))])) {10, Freshness}
13. believes(b,believes(a,[nb,goodkey(kab,a,b),believes(s,fresh(goodkey(kab,a,b))])) {12, 11, Nonce verification}
14. believes(b,believes(a,goodkey(kab,a,b))) {13, Belief}

1. sends(a,b,[encrypt(kbs,goodkey(kab,a,b)),encrypt(kab,combined(nb,[nb,goodkey(kab,a,b),believes(s,fresh(goodkey(kab,a,b))])))] {Step}
2. sees(b,[encrypt(kbs,goodkey(kab,a,b)),encrypt(kab,combined(nb,[nb,goodkey(kab,a,b),believes(s,fresh(goodkey(kab,a,b))])))] {1, Seeing}
3. believes(b,goodkey(kbs,b,s)) {Assumption}
4. sees(b,encrypt(kbs,goodkey(kab,a,b))) {2, Sight}
5. believes(b,goodkey(kbs,s,b)) {3, Bidirectionality}
6. sees(b,encrypt(kab,combined(nb,[nb,goodkey(kab,a,b),believes(s,fresh(goodkey(kab,a,b))])))) {2, Sight}
7. believes(b,oncesaid(s,goodkey(kab,a,b))) {5, 4, Message meaning}
8. sees(b,combined(nb,[nb,goodkey(kab,a,b),believes(s,fresh(goodkey(kab,a,b))])) {7, 6, Message decryption}
9. believes(b,secret(nb,a,b)) {Assumption}
10. believes(b,oncesaid(a,[nb,goodkey(kab,a,b),believes(s,fresh(goodkey(kab,a,b))])) {9, 8, Message meaning}
11. believes(b,fresh(nb)) {Assumption}
12. believes(b,fresh([nb,goodkey(kab,a,b),believes(s,fresh(goodkey(kab,a,b))])) {11, Freshness}
13. believes(b,believes(a,[nb,goodkey(kab,a,b),believes(s,fresh(goodkey(kab,a,b))])) {12, 10, Nonce verification}
14. believes(b,believes(a,believes(s,fresh(goodkey(kab,a,b))))) {13, Belief}
15. believes(b,jurisdiction(a,believes(s,fresh(goodkey(_1264312,a,b))))) {Assumption}
16. believes(b,believes(s,fresh(goodkey(kab,a,b))))) {15, 14, Jurisdiction}
17. believes(b,jurisdiction(s,fresh(goodkey(_1264448,a,b))))) {Assumption}
18. believes(b,oncesaid(a,goodkey(kab,a,b))) {10, Utterance}
19. believes(b,fresh(goodkey(kab,a,b))) {17, 16, Jurisdiction}
20. believes(b,believes(a,goodkey(kab,a,b))) {19, 18, Nonce verification}

yes

```

| ?- [user].
[Compiling user]
goal(believes(a, goodkey(kab, a, b))).
goal(believes(a, believes(b, na))).
goal(believes(b, goodkey(kab, a, b))).
goal(believes(b, believes(a, goodkey(kab, a, b)))).
[user compiled, cpu time used: 0.91 seconds]
[user loaded]

yes
| ?- chk_red(a, A, As).

no
| ?- chk_red(b, A, As).

no

```

B.5 The Needham-Schroeder Public-Key Protocol

Message 1 $A \rightarrow S : A, B$
 Message 2 $S \rightarrow A : \{K_b, B\}_{K_s^{-1}}$
 Message 3 $A \rightarrow B : \{N_a, A\}_{K_b}$
 Message 4 $B \rightarrow S : B, A$
 Message 5 $S \rightarrow B : \{K_a, A\}_{K_s^{-1}}$
 Message 6 $B \rightarrow A : \{N_a, N_b\}_{K_a}$
 Message 7 $A \rightarrow B : \{N_b\}_{K_b}$

B.5.1 Program Representation

% The Needham-Schroeder Public-Key Protocol

% Idealized Protocol

```

fact(1, sends(s, a, encrypt(inv(ks), public(kb, b))), reason([], 'Step')).
fact(2, sends(a, b, encrypt(kb, na)), reason([], 'Step')).
fact(3, sends(s, b, encrypt(inv(ks), public(ka, a))), reason([], 'Step')).
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Idealized Message 6 differs from the one given in [1]
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
fact(4, sends(b, a, encrypt(ka, combined(na, [na, secret(nb, a, b)]))), reason([], 'Step')).
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Idealized Message 7 differs from the one given in [1]
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

```
fact(5, sends(a,b,encrypt(kb,combined(nb,[nb, secret(na,a,b)],believes(b,secret(nb,a,b))))),
reason([], 'Step')).
```

```
% Assumptions
```

```
fact(6, believes(a, public(ka, a)), reason([], 'Assumption')).
fact(7, believes(a, public(ks, s)), reason([], 'Assumption')).
fact(8, believes(a, jurisdiction(s, public(_, b))), reason([], 'Assumption')).
fact(9, believes(a, fresh(na)), reason([], 'Assumption')).
fact(10, believes(a, secret(na, a, b)), reason([], 'Assumption')).
fact(11, believes(a, fresh(public(kb, b))), reason([], 'Assumption')).
fact(12, believes(b, public(kb, b)), reason([], 'Assumption')).
fact(13, believes(b, public(ks, s)), reason([], 'Assumption')).
fact(14, believes(b, jurisdiction(s, public(_, a))), reason([], 'Assumption')).
fact(15, believes(b, fresh(nb)), reason([], 'Assumption')).
fact(16, believes(b, secret(nb, a, b)), reason([], 'Assumption')).
fact(17, believes(b, fresh(public(ka, a))), reason([], 'Assumption')).
fact(18, believes(s, public(ka, a)), reason([], 'Assumption')).
fact(19, believes(s, public(kb, b)), reason([], 'Assumption')).
fact(20, believes(s, public(ks, s)), reason([], 'Assumption')).
```

B.5.2 Proofs

```
| ?- analyze(nspk).
```

```
Analyzed in 4 cycles
```

```
yes
```

```
| ?- explain_proof(believes(a, public(kb, b))).
1. sends(s,a,encrypt(inv(ks),public(kb,b))) {Step}
2. sees(a,encrypt(inv(ks),public(kb,b))) {1, Seeing}
3. believes(a,public(ks,s)) {Assumption}
4. believes(a,oncesaid(s,public(kb,b))) {3, 2, Message meaning}
5. believes(a,fresh(public(kb,b))) {Assumption}
6. believes(a,believes(s,public(kb,b))) {5, 4, Nonce verification}
7. believes(a,jurisdiction(s,public(_1258528,b))) {Assumption}
8. believes(a,public(kb,b)) {7, 6, Jurisdiction}
```

```
yes
```

The need to idealize Message 6 as $\{\langle N_a, A \stackrel{N_b}{\rightleftharpoons} B \rangle_{N_a}\}_{K_a}$ is highlighted by the following proof:

```
| ?- explain_proof(believes(a, believes(b, secret(nb, a, b)))).
1. sends(b,a,encrypt(ka,combined(na,[na,secret(nb,a,b)]))) {Step}
2. sees(a,encrypt(ka,combined(na,[na,secret(nb,a,b)]))) {1, Seeing}
3. believes(a,public(ka,a)) {Assumption}
4. believes(a,secret(na,a,b)) {Assumption}
5. sees(a,combined(na,[na,secret(nb,a,b)])) {3, 2, Message decryption}
```

```

6. believes(a,secret(na,b,a)) {4, Bidirectionality}
7. believes(a,fresh(na)) {Assumption}
8. believes(a,oncesaid(b,[na,secret(nb,a,b)])) {6, 5, Message meaning}
9. believes(a,fresh([na,secret(nb,a,b)])) {7, Freshness}
10. believes(a,believes(b,[na,secret(nb,a,b)])) {9, 8, Nonce verification}
11. believes(a,believes(b,secret(nb,a,b))) {10, Belief}

```

yes

```

| ?- explain_proof(believes(b, public(ka, a))).
1. sends(s,b,encrypt(inv(ks),public(ka,a))) {Step}
2. sees(b,encrypt(inv(ks),public(ka,a))) {1, Seeing}
3. believes(b,public(ks,s)) {Assumption}
4. believes(b,oncesaid(s,public(ka,a))) {3, 2, Message meaning}
5. believes(b,fresh(public(ka,a))) {Assumption}
6. believes(b,believes(s,public(ka,a))) {5, 4, Nonce verification}
7. believes(b,jurisdiction(s,public(_1258528,a))) {Assumption}
8. believes(b,public(ka,a)) {7, 6, Jurisdiction}

```

yes

The need to idealize Message 7 as $\{\langle N_b, A \stackrel{N_a}{\equiv} B, B \models A \stackrel{N_b}{\equiv} B \rangle_{N_b}\}_{K_b}$ is highlighted by the following proof:

```

| ?- explain_proof(believes(b, believes(a, secret(na, a, b)))).
1. sends(a,b,encrypt(kb,combined(nb,[nb,secret(na,a,b),believes(b,secret(nb,a,b)])))) {Step}
2. sees(b,encrypt(kb,combined(nb,[nb,secret(na,a,b),believes(b,secret(nb,a,b)])))) {1, Seeing}
3. believes(b,public(kb,b)) {Assumption}
4. sees(b,combined(nb,[nb,secret(na,a,b),believes(b,secret(nb,a,b)]))) {3, 2, Message decryption}
5. believes(b,secret(nb,a,b)) {Assumption}
6. believes(b,fresh(nb)) {Assumption}
7. believes(b,oncesaid(a,[nb,secret(na,a,b),believes(b,secret(nb,a,b)]))) {5, 4, Message meaning}
8. believes(b,fresh([nb,secret(na,a,b),believes(b,secret(nb,a,b)]))) {6, Freshness}
9. believes(b,believes(a,[nb,secret(na,a,b),believes(b,secret(nb,a,b)]))) {8, 7, Nonce verification}
10. believes(b,believes(a,secret(na,a,b))) {9, Belief}

```

yes

```

| ?- [user].
[Compiling user]
goal(believes(a, public(kb, b))).
goal(believes(a, believes(b, secret(nb, a, b)))).
goal(believes(b, public(ka, a))).
goal(believes(b, believes(a, secret(na, a, b)))).
[user compiled, cpu time used: 0.28003 seconds]
[user loaded]

```

```

yes
| ?- chk_red(a, A, As).

no
| ?- chk_red(b, A, As).

no

```

B.6 The CCITT.X509 Protocol

Message 1 $A \rightarrow B : A, \{T_a, N_a, B, X_a, \{Y_a\}_{K_b}\}_{K_a^{-1}}$
 Message 2 $B \rightarrow A : B, \{T_b, N_b, A, N_a, X_b, \{Y_b\}_{K_a}\}_{K_b^{-1}}$
 Message 3 $A \rightarrow B : A, \{N_b\}_{K_a^{-1}}$

B.6.1 Program Representation

```

% The CCITT X.500 Protocol

% Idealized Protocol
fact(1, sends(a, b, encrypt(inv(ka), [ta, na, xa, encrypt(kb, ya)])), reason([], 'Step')).
fact(2, sends(b, a, encrypt(inv(kb), [tb, nb, na, xb, encrypt(ka, yb)])), reason([], 'Step')).
fact(3, sends(a, b, encrypt(inv(ka), nb)), reason([], 'Step')).

% Assumptions
fact(4, believes(a, public(ka, a)), reason([], 'Assumption')).
fact(5, believes(a, public(kb, b)), reason([], 'Assumption')).
fact(6, believes(a, fresh(na)), reason([], 'Assumption')).
fact(7, believes(a, fresh(tb)), reason([], 'Assumption')).
fact(8, believes(b, public(kb, b)), reason([], 'Assumption')).
fact(9, believes(b, public(ka, a)), reason([], 'Assumption')).
fact(10, believes(b, fresh(nb)), reason([], 'Assumption')).
fact(11, believes(b, fresh(ta)), reason([], 'Assumption')).

```

B.6.2 Proofs

```

| ?- analyze(ccitt).
Analyzed in 3 cycles

yes
| ?- explain_proof(believes(a, believes(b, xb))).
1. sends(b,a,encrypt(inv(kb),[tb,nb,na,xb,encrypt(ka,yb)])) {Step}
2. sees(a,encrypt(inv(kb),[tb,nb,na,xb,encrypt(ka,yb)])) {1, Seeing}
3. believes(a,public(kb,b)) {Assumption}
4. believes(a,fresh(na)) {Assumption}
5. believes(a,oncesaid(b,[tb,nb,na,xb,encrypt(ka,yb)])) {3, 2, Message meaning}
6. believes(a,fresh([tb,nb,na,xb,encrypt(ka,yb)])) {4, Freshness}

```

```

7. believes(a,believes(b,[tb,nb,na,xb,encrypt(ka,yb)])) {6, 5, Nonce verification}
8. believes(a,believes(b,xb)) {7, Belief}

```

```

1. sends(b,a,encrypt(inv(kb),[tb,nb,na,xb,encrypt(ka,yb)])) {Step}
2. sees(a,encrypt(inv(kb),[tb,nb,na,xb,encrypt(ka,yb)])) {1, Seeing}
3. believes(a,public(kb,b)) {Assumption}
4. believes(a,fresh(tb)) {Assumption}
5. believes(a,oncesaid(b,[tb,nb,na,xb,encrypt(ka,yb)])) {3, 2, Message meaning}
6. believes(a,fresh([tb,nb,na,xb,encrypt(ka,yb)])) {4, Freshness}
7. believes(a,believes(b,[tb,nb,na,xb,encrypt(ka,yb)])) {6, 5, Nonce verification}
8. believes(a,believes(b,xb)) {7, Belief}

```

yes

```

| ?- explain_proof(sees(a, yb)).
1. sends(b,a,encrypt(inv(kb),[tb,nb,na,xb,encrypt(ka,yb)])) {Step}
2. sees(a,encrypt(inv(kb),[tb,nb,na,xb,encrypt(ka,yb)])) {1, Seeing}
3. believes(a,public(kb,b)) {Assumption}
4. sees(a,[tb,nb,na,xb,encrypt(ka,yb)]) {3, 2, Message decryption}
5. sees(a,encrypt(ka,yb)) {4, Sight}
6. believes(a,public(ka,a)) {Assumption}
7. sees(a,yb) {6, 5, Message decryption}

```

yes

```

| ?- explain_proof(believes(b, believes(a, nb))).
1. sends(a,b,encrypt(inv(ka),nb)) {Step}
2. sees(b,encrypt(inv(ka),nb)) {1, Seeing}
3. believes(b,public(ka,a)) {Assumption}
4. believes(b,oncesaid(a,nb)) {3, 2, Message meaning}
5. believes(b,fresh(nb)) {Assumption}
6. believes(b,believes(a,nb)) {5, 4, Nonce verification}

```

yes

```

| ?- explain_proof(sees(b, ya)).
1. sends(a,b,encrypt(inv(ka),[ta,na,xa,encrypt(kb,ya)])) {Step}
2. sees(b,encrypt(inv(ka),[ta,na,xa,encrypt(kb,ya)])) {1, Seeing}
3. believes(b,public(ka,a)) {Assumption}
4. sees(b,[ta,na,xa,encrypt(kb,ya)]) {3, 2, Message decryption}
5. sees(b,encrypt(kb,ya)) {4, Sight}
6. believes(b,public(kb,b)) {Assumption}
7. sees(b,ya) {6, 5, Message decryption}

```

yes

```

| ?- [user].
[Compiling user]
goal(believes(a, believes(b, xb))).
goal(sees(a, yb)).

```

```

goal(sees(b, ya)).
goal(believes(b, believes(a, nb))).
goal(believes(b, believes(a, xa))).
[user compiled, cpu time used: 0.831 seconds]
[user loaded]

yes
| ?- chk_red(a, A, As).

A = believes(a,fresh(na))
As = [believes(a,public(ka,a)),believes(a,public(kb,b)),believes(a,fresh(tb))];

A = believes(a,fresh(tb))
As = [believes(a,public(ka,a)),believes(a,public(kb,b)),believes(a,fresh(na))];

no
| ?- chk_red(b, A, As).

no

```


Appendix C

BAN Logic Analyzer Program

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% File      : ban.P
%
% Author    : Anish Mathuria
%
% Description : This module contains predicates for the BAN Logic
%               protocol analyzer program written in XSB Prolog.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

:- import member/2, append/3, reverse/2, ith/3, select/3 from basics.

% analyze(Protocol)
% Top-level predicate. It's argument is the full path name of the file
% defining the protocol to be analyzed.

analyze(Protocol) :-
    abolish(flag, 2),
    load_dyn(Protocol),
    findall(Index, fact(Index, _, _), Indices),
    max(Indices, MaxIndex),
    asserta(flag(count, MaxIndex)),
    asserta(flag(new_fact, no)),
    setup,
    forward(0).

% setup
```

```

setup :-
    fact(I, sends(_, Q, X), _),
    flag(count, CurrIndex),
    NewIndex is CurrIndex + 1,
    assertz(fact(NewIndex, sees(Q, X), reason([I], 'Seeing'))),
    set_flag(count, NewIndex),
    fail.
setup.

% forward(Cycle) - Main Driver
% Repeatedly cycles through the rules defined for r/0. It
% succeeds when no new statements can be derived. The
% argument Cycle is incremented with every rule cycle
% performed.
forward(Cycle) :-
    Cycle > 0,
    done,
    write('Analyzed in '), write(Cycle), write(' cycles'), nl.

forward(Cycle) :-
    apply_rules,
    NextCycle is Cycle + 1,
    forward(NextCycle).

% done/0 checks the status of the global flag new_fact. It succeeds when
% new_fact is no. This corresponds to the case when no fact is added in
% the rule cycle. Otherwise when new_fact is yes it fails and resets
% new_flag to no.
done :-
    flag(new_fact, no).
done :-
    flag(new_fact, yes),
    set_flag(new_fact, no),
    fail.

% apply_rules - Applies all inference rules
% Applies inference rules to the facts in the database
% and stores new formulae derived. Always succeeds.
apply_rules :-

```

```

(
    sight(Conclusion, Reason)
;
    message_decryption(Conclusion, Reason)
;
    message_meaning(Conclusion, Reason)
;
    freshness(Conclusion, Reason)
;
    nonce_verification(Conclusion, Reason)
;
    belief(Conclusion, Reason)
;
    utterance(Conclusion, Reason)
;
    jurisdiction(Conclusion, Reason)
;
    bidirectionality(Conclusion, Reason)
),
    addfact(Conclusion, Reason),
    fail.
apply_rules.

% Inference rules

% Message-meaning rules
% For shared keys
message_meaning(believes(P, oncesaid(Q, X)),
    reason([I1,I2], 'Message meaning')) :-
    . fact(I1, believes(P, goodkey(K, Q, P)), _),
      fact(I2, sees(P, encrypt(K, X)), _).

% For Public keys
message_meaning(believes(P, oncesaid(Q, X)),
    reason([I1,I2], 'Message meaning')) :-
    fact(I1, believes(P, public(K, Q)), _),
    fact(I2, sees(P, encrypt(inv(K), X)), _).

% For shared secrets
message_meaning(believes(P, oncesaid(Q, X)),

```

```

        reason([I1,I2], 'Message meaning')) :-
fact(I1, believes(P, secret(Y, Q, P)), _),
fact(I2, sees(P, combined(Y, X)), _).

% Nonce-verification rule
nonce_verification(believes(P, believes(Q, X)),
        reason([I1,I2], 'Nonce verification')) :-
fact(I1, believes(P, fresh(X)), _),
fact(I2, believes(P, oncesaid(Q, X)), _).

% Jurisdiction rule
jurisdiction(believes(P, X), reason([I1, I2], 'Jurisdiction')) :-
fact(I1, believes(P, jurisdiction(Q, X)), _),
fact(I2, believes(P, believes(Q, X)), _).

% Belief rules
belief(believes(P, M), reason([I], 'Belief')) :-
fact(I, believes(P, L), _),
member(M, L).
belief(believes(P, believes(Q, M)), reason([I], 'Belief')) :-
fact(I, believes(P, believes(Q, L)), _),
member(M, L).

% Utterance rule
utterance(believes(P, oncesaid(Q, M)), reason([I], 'Utterance')) :-
fact(I, believes(P, oncesaid(Q, L)), _),
member(M, L).

% Sight rules
sight(sees(P, M), reason([I], 'Sight')) :-
fact(I, sees(P, L), _),
member(M, L).
sight(sees(P, X), reason([I], 'Sight')) :-
. fact(I, sees(P, combined(_, X)), _).

% Message decryption rules
% For shared keys
message_decryption(sees(P, X), reason([I1, I2], 'Message decryption')) :-
fact(I1, believes(P, goodkey(K, _, P)), _),
fact(I2, sees(P, encrypt(K, X)), _).

```

```

% For public keys
message_decryption(sees(P, X), reason([I1, I2], 'Message decryption')) :-
    fact(I1, believes(P, public(K, P)), _),
    fact(I2, sees(P, encrypt(K, X)), _).
message_decryption(sees(P, X), reason([I1, I2], 'Message decryption')) :-
    fact(I1, believes(P, public(K, _)), _),
    fact(I2, sees(P, encrypt(inv(K), X)), _).

% Uncertified keys
message_decryption(sees(P, X), reason([I1, I2], 'Message decryption')) :-
    fact(I1, believes(P, oncesaid(_, goodkey(K, _, P))), _),
    fact(I2, sees(P, encrypt(K, X)), _).

% Freshness rule
freshness(believes(P, fresh(X)), reason([I], 'Freshness')):-
    fact(I, believes(P, fresh(M)), _),
    fact(_, believes(P, oncesaid(_, X)), _),
    member(M, X).

% Bidirectionality rules
% For shared keys
bidirectionality(believes(P, goodkey(K, R, Q)),
    reason([I], 'Bidirectionality')) :-
    fact(I, believes(P, goodkey(K, Q, R)), _).
bidirectionality(believes(P, believes(Q, goodkey(K, R, S))),
    reason([I], 'Bidirectionality')) :-
    fact(I, believes(P, believes(Q, goodkey(K, S, R))), _).

% For shared secrets
bidirectionality(believes(P, secret(X, R, Q)),
    reason([I], 'Bidirectionality')) :-
    fact(I, believes(P, secret(X, Q, R)), _).
bidirectionality(believes(P, believes(Q, secret(X, S, R))),
    reason([I], 'Bidirectionality')) :-
    fact(I, believes(P, believes(Q, secret(X, R, S))), _).

% Maintains derived facts
addfact(Formula, reason(PremIs, Rule)) :-
    check(Formula, reason(PremIs, Rule)),

```

```

    flag(count, CurrIndex),
    NewIndex is CurrIndex + 1,
    assertz(fact(NewIndex, Formula, reason(PremIs, Rule))),
    set_flag(count, NewIndex),
    set_flag(new_fact, yes).

% set_flag(Name, Val) - Maintains globals
% Sets the global flag Name to the value Val.
set_flag(Name, Val) :-
    nonvar(Name),
    retract(flag(Name, _)),
    !,
    asserta(flag(Name, Val)).
set_flag(Name, Val) :-
    nonvar(Name),
    asserta(flag(Name, Val)).

% check(Formula, Reason) - Loop checker
% Succeeds if insertion of the fact representing the derivation
% of the statement Formula whose derivation information is in
% Reason would lead to an infinite loop; fails otherwise.

% Inference already made
check(Formula, Reason) :-
    fact(_, Formula, Reason),
    !,
    fail.

% Check if the inferred statement Formula was used earlier in
% deriving a premise in PremIs
check(Formula, reason(PremIs, _)) :-
    findall(Index, fact(Index, Formula, _), Indices),
    Indices \= [],
    !,
    checkall(PremIs, Indices).

% New derivation
check(_, _).

checkall([], _).

```

```

checkall([PremIndex|PremIndices], Indices) :-
    proof(PremIndex, ProofOfPrem),
    intersection(Indices, [PremIndex|ProofOfPrem], []),
    checkall(PremIndices, Indices).

% explain_proof(Goal) - Proof Explanation
% Explains all proofs of statement Goal.
explain_proof(Goal) :-
    build_proof(Goal, Proof),
    write_proof(Proof),
    nl,
    fail.
explain_proof(_).

% build_proof(Goal, Proof) - Builds index list of proofs
% The list Proof represents a minimal proof of statement Goal.
build_proof(Goal, Proof) :-
    fact(Index, Goal, _),
    proof(Index, ProofDup),
    remdup([Index|ProofDup], RevProof),
    reverse(RevProof, Proof).

% proof(Index, ProofDup) - Traverses database
% Traverses the database to build the list ProofDup containing
% the indices of the facts used in a derivation of fact with
% index Index.
proof(Index, ProofDup) :-
    fact(Index, _, reason(Premises, _)),
    append(Premises, ProofOfPremises, ProofDup),
    prooflist(Premises, ProofOfPremises).

prooflist([], []).
prooflist([Premise|Premises], TotalProof) :-
    proof(Premise, ProofOfPremise),
    prooflist(Premises, ProofOfPremises),
    append(ProofOfPremise, ProofOfPremises, TotalProof).

% write_proof(Proof) - Prints proof
% Prints a formatted proof from the list Proof
write_proof(Proof) :-

```

```

write_proof(Proof, Proof).

write_proof([], _).
write_proof([Index|Indices], Proof) :-
    fact(Index, Formula, reason(Premises, Rule)),
    ith(StepNo, Proof, Index),
    write(StepNo), write(' '),
    write(' '), write(Formula), tab(2),
    write('{'), write_list(Premises, Proof), write(Rule),
    write('}'), nl,
    write_proof(Indices, Proof).

% write_list(Indices, Proof) - Prints justification of a step
% Prints step numbers of the premises in the list Indices
% using the list Proof as a reference.
write_list([], _).
write_list([Index|Indices], ProofIs) :-
    ith(StepNo, ProofIs, Index),
    write(StepNo), write(', '),
    write_list(Indices, ProofIs).

% chk_red(P, RedAssumption, MinAssumptions)
% Identifies redundant assumptions
% The list MinAssumptions is the minimal set of assumptions
% excluding a redundant assumption RedAssumption of principal P.
chk_red(P, RedAssumption, MinAssumptions) :-
    findall(Assumption, is_ass(P, Assumption, _), Assumptions),
    is_ass(P, RedAssumption, Index),
    findall(Goal, (goal(Goal), arg(1, Goal, P)), Goals),
    Goals \= [],
    chk_allfmla(Index, Goals),
    select(RedAssumption, Assumptions, MinAssumptions).

% is_ass(P, Assumption, Index) - Gets assumptions
% Index is the index of the database fact representing
% the assumption Assumption of principal P.
is_ass(P, Assumption, Index) :-
    fact(Index, Assumption, reason([], 'Assumption')),
    arg(1, Assumption, P).

```



```

% chk_allfmla(Index, Formulae) - Checks proofs of all formulae
chk_allfmla(_, []).

chk_allfmla(Index, [Formula|Formulae]) :-
    chk_fmlla(Index, Formula),
    chk_allfmla(Index, Formulae).

% chk_fmlla(Index, Formula) - Checks all proofs of a formula
chk_fmlla(Index, Formula) :-
    findall(Proof, build_proof(Formula, Proof), Proofs),
    chk_proof(Index, Proofs).

chk_proof(Index, [Proof|_]) :-
    not(member(Index, Proof)),
    !.
chk_proof(Index, [_|Proofs]) :-
    chk_proof(Index, Proofs).

% Miscellaneous list processing predicates

% max(List, Max) - Finds list maximum
% Max is the largest number in the list List
max([Max], Max).
max([Head|Tail], Max) :-
    max(Tail, MaxTail),
    Head >= MaxTail,
    !,
    Max = Head.
max([_|Tail], Max) :-
    max(Tail, Max).

% remove_dups(DupList, UniqList) - Removes duplicates
% Each element of the list DupList occurs
% in the list UniqList only once.
remove_dups([], []).
remove_dups([Head|Tail], UniqList) :-
    member(Head, Tail),
    !,
    remove_dups(Tail, UniqList).
remove_dups([Head|Tail], [Head|UniqTail]) :-
    remove_dups(Tail, UniqTail).

```

```

% intersection(List1, List2, List3) - Set intersection
% The list List3 is the intersection of the
% lists List1 and List2.
intersection([], _, []).
intersection([Head1|Tail1], List2, List3) :-
    member(Head1, List2),
    !,
    List3 = [Head1|Tail3],
    intersection(Tail1, List2, Tail3).
intersection([_|Tail1], List2, List3) :-
    intersection(Tail1, List2, List3).

```

